MICROINFORMÁTICA

MICROS DE LÓGICA SINCLAIR®

UM GUIA BÁSICO

Campbell · Siminoff · Yates

LTC LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A.

MICROS DE LÓGICA SINCLAIR®

UM GUIA BÁSICO

SÉRIE "MICROINFORMÁTICA"

Coordenação Técnica:
ANTÔNIO ROBERTO RAMOS NOGUEIRA

CONHEÇA TAMBÉM

BACK	— CP/M Sistema Operacional para Microcomputadores
BARBUTO	— 35 Programas BASIC para Microcomputadores
BASTOS	— Programação COBOL
BIANCHI	— Introdução aos Microcomputadores
BIANCHI/BEZERRA	— Microcomputadores — Arquitetura, Projeto e Programação
BORGES	— BASIC — Aplicações Comerciais
BOSCH	— COBOL — Fundamentos e Aplicações
CARDOSO	— Programação Estruturada em COBOL
CORDEIRO PAULO	— COBOL — Técnicas e Dispositivos Especiais
COUCEIRO/BARRENECHA	— Sistemas de Gerência de Banco de Dados Distribuídos
DIAS	— O Sistema de Informação e a Empresa
DIAS/GAZZANEO	— Projetos de Sistemas de Processamento de Dados
EADIE	— Minicomputadores — Teoria e Prática
FURTADO/PASSOS	— Introdução à Programação com PL/1
GANE/SARSON	— Análise Estruturada de Sistemas
GRILLO	— Programação Estruturada com FORTRAN
GUSMAN/VASCONCELLOS	— Fluxogramas e Programação COBOL
HELT/BONFIM/MARTINS	— Estatística em Microcomputadores
KATZAN	— Segurança de Dados em Computação
KUGLER/FERNANDES	— Planejamento e Controle de Sistemas de Informação
McNITT	— Simulação em BASIC
MAYNARD	— Programação Modular
McCALEB	— A Microinformática na Empresa
PACITTI	— Fortran Monitor .
PACITTI/ATKINSON	— Programação e Métodos Computacionais vols. 1 e 2
PEREIRA	— Computes, Grilo!
PINHEIRO/MACIEL/MOREIRA	— Transcrição de Dados
PRAÇA	— COBOL para Micros
PRADO	— Administração de Projetos com PERT/CPM
PRATES	— BASIC Aplicado — Um Enfoque Profissional
PUCCINI	— Introdução à Programação Linear
ROYO DOS SANTOS	— Processamento de Dados
SÁ	— Programação PL/1
SAPIRA/NESANELOVICZ	— Introdução à Linguagem APL
SILVA/HEIBEL	— VISICALC/CIRCALC/PROCALC
SOUZA	— Introdução à Linguagem BASIC
STRACK	— GPSS — Modelagem e Simulação de Sistemas
STRACK	— Sistemas de Processamento Distribuído
SUCESU	— Dicionário de Informática
TAROUCO	— Redes de Comunicação de Dados
VASCONCELLOS	— Computadores Eletrônicos e Processamento
VASCONCELLOS/SZERMAN	— O Centro de Processamento de Dados
VON STAA	— Engenharia de Programas
WOOLDRIDGE/LONDON	— O Computador e o Executivo
ZIMMERMANN	— Linguagem de Programação APL

CARTÕES DE REFERÊNCIA

DICHAUNE	— Wordstar
HADA	— Visicalc
KLEIBER	— COBOL para Micros
PRATES	— CP/M
PRATES	— MS-DOS
SILVA/HEIBEL	— SUPERCALC

MICROS DE LÓGICA SINCLAIR®

UM GUIA BÁSICO

Joe Campbell
Jonathan D. Siminoff
Jean Yates

Tradução de Josir Cardoso Gomes

Rio de Janeiro-RJ • São Paulo-SP

GRANDE QUANTIDADE DE INFORMAÇÕES A SEU ALCANCE

A menos de 30 anos o UNIVAC I, o primeiro computador eletrônico produzido em série, custava um milhão de dólares e ocupava uma sala imensa. Hoje o Timex/Sinclair 1000[1] é vendido por menos de US$ 100, encaixa-se em sua pasta ou maleta e é tão poderoso quanto o UNIVAC e muito mais fácil de ser usado — se você souber como.

Agora, este manual muito fácil de ser entendido irá ensinar você a usar o seu T/S 1000 nos negócios ou em casa, para seu benefício, economia, conforto, eficiência, educação e diversão com seus jogos. Você descobrirá como ele funciona, qual a função de suas diferentes partes, como programá-lo e como expandir suas potencialidades com os periféricos. E isso é apenas uma pequena amostra do que você vai descobrir e de como se divertirá com este manual indispensável à nova era dos computadores.

---

[1] O Timex/Sinclair 1000 é mais uma das versões do ZX-81, o microcomputador revolucionário feito por Clive Sinclair e que lhe rendeu milhões de dólares e até um título de nobreza. Aqui no Brasil, ele é representado pelos TKS (82-C, 83 e 85)®, CP-200® e outros caracterizados pela denominação de micros de lógica Sinclair. Logo, sempre que me referir aos T/S 1000, estarei me referindo também a todos os "Microcomputadores de Lógica Sinclair."

SUMÁRIO

APÊNDICES

PREFÁCIO

Você já ouviu falar que a revolução da era dos computadores está a caminho. Agora ela já chegou e o seu T/S 1000 dá a você a oportunidade de participar dela. Este livro representa sua admissão no apressado mundo da tecnologia. Até bem poucos anos atrás, os computadores eram um campo de conhecimento dos matemáticos especializados que treinavam anos, antes que pudessem trabalhar com máquina do tamanho de um armazém. Hoje, qualquer um que tenha tempo para ler este livro pode aprender a usar seu computador pessoal T/S 1000, o primeiro vendido por menos de US$ 100, e tão pequeno que você pode levá-lo aonde quiser.

Use o TS/1000 para controlar seu talão de cheques, calcular seus impostos ou avaliar que banco oferecerá a você mais vantagens para efetuar um investimento. É tanto uma ferramenta educacional quanto um brinquedo divertido. Ligue-o em sua TV e divirta-se com os seus jogos. Faça os seus próprios videogames — este livro mos-

trará como. Mas, o mais importante é que o T/S 1000 vai fazer de você um cientista em potencial, capaz de usar um equipamento maior e mais sofisticado. O T/S 1000 usa o *BASIC*, a linguagem de programação mais popular do mundo. Após aprender o *BASIC* através deste livro, você estará apto para programar computadores para qualquer propósito. Este é um computador que *todos* podem usar.

O Capítulo 1 explica de que são feitos e como funcionam os computadores. O Capítulo 2 inicia você na carreira de programador; nele você aprenderá o elementar dc *BASIC* e como escrever e depurando programas.

No momento em que você terminar o Capítulo 3 e seu primeiro programa estiver pronto, você poderá se considerar um veterano em computação. Para aprender a armazenar seu programa em fita, leia o Capítulo 5. Você é ou pretende ser um jogador de videogames? Então pule para os Capítulos 7 e 8. Você planeja usar o seu computador em casa? O Capítulo 10 cobre seu uso no lar, desde o controle do talão de cheques ao cálculo dos pagamentos dos empréstimos. Talvez você esteja mais interessado na melhor maneira de utilizar o seu computador no escritório; veja o Capítulo 11, para maiores detalhes.

Finalmente, existem os apêndices — guias sobre os acessórios do computador, as partes internas e os recursos para você saber mais sobre ele. Há também um glossário e outro apêndice que explica todos os mistérios do teclado. Um apêndice final traz uma lista dos códigos próprios do T/S 1000.

Você está cansado de ficar nas preliminares? Quer participar do século XXI? *Então vire a página...*

1

O INÍCIO

Os computadores são práticos, quietos e obedientes. Eles farão exatamente o que você mandar, contanto que seja fisicamente possível e que compreendam as instruções dadas por você. Mas primeiro você precisa aprender sua linguagem especial.

Seu T/S 1000 é uma excelente ferramenta para ensinar você como usar um computador. Por esta razão, este livro foi planejado para ser usado com o seu T/S 1000 diante de você. Para cada ponto dado, um exemplo é apresentado; experimente todos eles. Usar um computador é como andar de bicicleta, nadar, cozinhar ou jogar futebol. Alguém pode iniciá-lo dando-lhe a orientação certa, mas você realmente só pode aprender praticando.

Como Instalar o seu Computador[2]

Desocupe uma mesa de bom tamanho e coloque sobre ela sua televisão. Uma televisão em preto e branco é melhor, mas qualquer tipo serve. Quando você retirar o seu T/S 1000 novo, de dentro da caixa, você encontrará o seguinte:

O computador. Como você deve ter avaliado, ele é o objeto com o teclado em sua parte superior. Possui quatro conectores do lado esquerdo e que são chamados TV, EAR, MIC e 9V DC.

Uma fonte de alimentação. É uma caixa pequena, quadrada, com um cabo de alimentação ligado a ela. Ligue a fonte a uma tomada ou a uma extensão e o final do cabo ao conector do computador chamado 9V DC. Não tenha receio de ligá-lo no conector errado ou de tocá-lo acidentalmente; nenhum dano irá ocorrer mas o cabo deve ser ligado no conector certo, para que o computador possa funcionar.

Uma caixa de conexão da antena. É uma pequena caixa de metal de 3 x 2 x 1 polegadas com um fio duplo,

---

[2] No Brasil cada fabricante constrói um computador à sua maneira, isto é, não há um padrão predefinido na elaboração de micros da lógica Sinclair. Todos têm particularidades que só o manual que acompanha o equipamento pode mostrar. Este livro apresenta um roteiro geral, mas que só serve especificamente para o T/S 1000, pois nos demais equipamentos os *plugs*, os soquetes e outras peças certamente não estarão indicados corretamente.

curto e plano vindo de um dos lados e uma chave indicando *COMPUTER/TV.* Posicione a chave para a posição *COMPUTER.* Agora,observe a parte de trás da sua TV. Ela deve conter duas ou quatro conexões para ligar as antenas. Se a televisão tiver antena interna, as conexões provavelmente estarão perto dela. A maioria das televisões tem dois pares de conexões, um par para UHF e outro para VHF. A conexão de VHF, usada para os canais 2 a 13, são regularmente o par inferior. Desaparafuse o conector de antena VHF levemente, coloque a parte em forma de *U,* do final do cabo plano, abaixo da conexão e aperte-o. Não importa em qual parafuso você vai encaixar o conector em forma de *U,* contanto que encaixe em parafusos diferentes e que ambos sejam de VHF. Se existirem apenas dois conectores, eles serão provavelmente de VHF; logo, também podem ser usados. Se sua televisão tiver uma chave para antena interna/externa conectada junto aos conectores de antena, coloque-a na posição externa.

Um cabo conector de vídeo. É um cabo de quase 4 pés de comprimento com um plug em cada ponta. Encaixe qualquer plug no soquete *TV,* no lado esquerdo do computador, e a outra extremidade no conector da caixa de conexão da antena. Você verá um soquete de um lado que se projeta um pouco, para acomodar o cabo conector de vídeo.

Um livro. O livro pode servir como seu manual de referência. Não é, entretanto, um texto ideal para aprender a

usar o seu T/S 1000. O livro que você está lendo agora foi escrito especialmente para este propósito.

Um cabo de dois pés com plugs em cada extremidade. É usado quando conectado com o seu computador e o gravador cassete. Coloque-o de lado, até você chegar ao Capítulo 5.

Seu computador está agora completamente instalado. Ligue sua televisão e espere até que apareça a imagem. Coloque a sua TV no Canal 2 ou 3, ou qualquer um que tenha uma estação fraca em sua região. Diminua totalmente o volume.

Na parte de baixo do computador você achará uma chave com as posições *ch 2* e *ch 3*. Escolha a posição que combine com o canal que você escolheu.

A tela de sua televisão deve agora ficar branca exceto no canto inferior esquerdo onde você deve ver um pequeno quadrado preto com um K dentro (se você estiver usando uma televisão colorida, deve haver alguma cor na imagem. Se isto não for de seu agrado, desligue o controle de cor). Se você não vir o quadrado ou se a imagem não estiver estável, ajuste os controles de sua TV para melhorá-la. Se a imagem ainda não estiver satisfatória, tente mudar para outro canal (se você estiver no Canal 2, tente o 3 e vice-versa). Aqui está uma lista dos possíveis problemas:

1. Tenha certeza de que a caixa de conexão da antena está na posição *COMPUTER*.

2. Se sua televisão tiver uma chave de antena interna/externa, tente ambas as posições.
3. Tente desconectar a antena normal da televisão.
4. Se a tela da televisão estiver acesa, mas o K não aparecer, tente ajustar o controle vertical e horizontal de sua televisão. É possível que o K esteja escondido abaixo da tela. Caso sua TV receba bem as estações locais, elas irão funcionar com o computador.
5. Se você tiver uma televisão a cabo, tente desconectar o cabo. Televisões assim são algumas vezes conectadas internamente; logo é possível que você tenha que utilizar uma TV que não esteja conectada para cabo.
6. Os dois plugs no final do cabo de vídeo estão bem conectados em seus soquetes? Eles devem estar suficientemente seguros para não se soltarem acidentalmente.
7. O computador está ligado? Verifique se o cabo condutor da fonte de alimentação está conectado corretamente. Tenha certeza que a fonte está ligada a uma tomada que funcione.
8. Finalmente, tente outra televisão. Se você tiver uma de boa qualidade, pequena, em preto e branco e portátil, esta é a melhor.
9. Se nada funcionar e o computador não funciona com nenhuma televisão que você tentou, devolva-o à loja e peça ao vendedor para ele próprio tentar. Você pode ter recebido uma unidade defeituosa.

Eu vou presumir que você tenha chegado a este ponto. Sua televisão agora tem uma imagem nítida e estável com o K no canto inferior esquerdo. Caso o seu computador venha a necessitar de imagens diferentes, como o seu programa favorito de TV, ajuste o brilho e o contraste para combinar com o seu gosto.

Agora está na hora de pôr o seu computador para trabalhar!

2

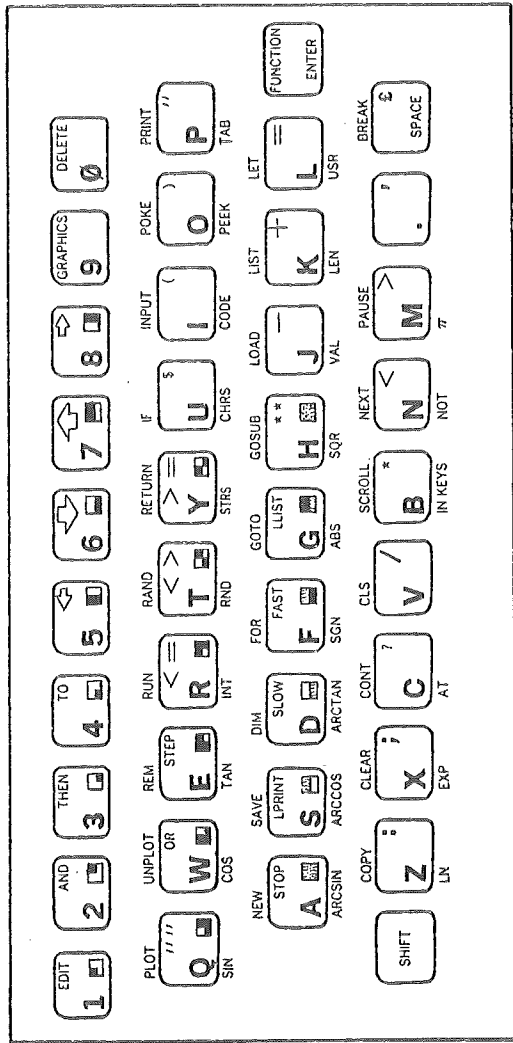
COMO USAR O TIMEX/SINCLAIR 1000

O quadrado preto em sua tela chama-se *cursor.* Ele indica que o computador está esperando que você lhe mande um comando e também indica onde o próximo caractere será impresso na tela. Olhe para o teclado impresso em seu computador. Se você já tiver usado uma máquina de escrever, a posição das teclas lhe será familiar. Pressionando a letra P, a palavra **PRINT** será impressa, ao invés do P; e o K dentro do cursor muda para um L. Embora o teclado se pareça com o de uma máquina de escrever, em miniatura, as teclas têm um funcionamento diferente. A maioria das teclas tem palavras impressas acima e abaixo de si e tem palavras ou símbolos junto com números ou letras impressos sobre a tecla.

Quando o K aparece na última linha da tela, o computador está pedindo a você que entre com uma *palavra-chave* que é a parte mais importante de um comando. Quando o computador espera uma palavra-chave, ao se digitar uma das teclas de letras (de A a Z), ele imprimirá toda a palavra-chave escrita acima da tecla. Logo, assim que você digitar a letra P, o **PRINT** aparece. Tente digitar um P, agora que o cursor L está na tela. Desta vez um P aparece na TV. O cursor L significa que o computador espera algo além da palavra-chave. As palavras-chave escritas acima das teclas serão impressas somente se o cursor K estiver na tela. A propósito, o T/S 1000 foi feito para imprimir apenas letras maiúsculas.

Toda tecla tem, dentro, algo impresso em vermelho. Quando você pressionar a tecla **SHIFT** e, simultaneamente, um pouco de qualquer outra tecla, seu símbolo em vermelho será impresso na tela.

A palavra em vermelho da tecla do canto superior direito do teclado (a tecla 0) é o **DELETE.** Pressione a tecla **SHIFT** e ao mesmo tempo a **DELETE.** A tecla P que você tinha digitado anteriormente desaparece. **DELETE SHIFT 0** apaga a letra ou comando imediatamente à esquerda do cursor. Pressione **DELETE** novamente e a palavra-chave **PRINT** desaparece. Se você já fez tudo isso, você agora voltou para o início da linha e o cursor, mais uma vez, é o K .Tecle P novamente e o **PRINT** aparecerá de novo. O cursor agora é o L .

Mantenha o **SHIFT** pressionando e tecle P mais uma vez. As aspas (" ") escritas em vermelho na tecla P apa-

O teclado do T/S 1000

recem no vídeo. Solte o **SHIFT**. Digite a palavra ALO. Observe que agora, quando você pressiona as teclas alfabéticas, as letras aparecerão. Tecle **SHIFT** junto com o P, de novo. Outras aspas aparecem. A linha deve parecer assim:

PRINT - "ALO"

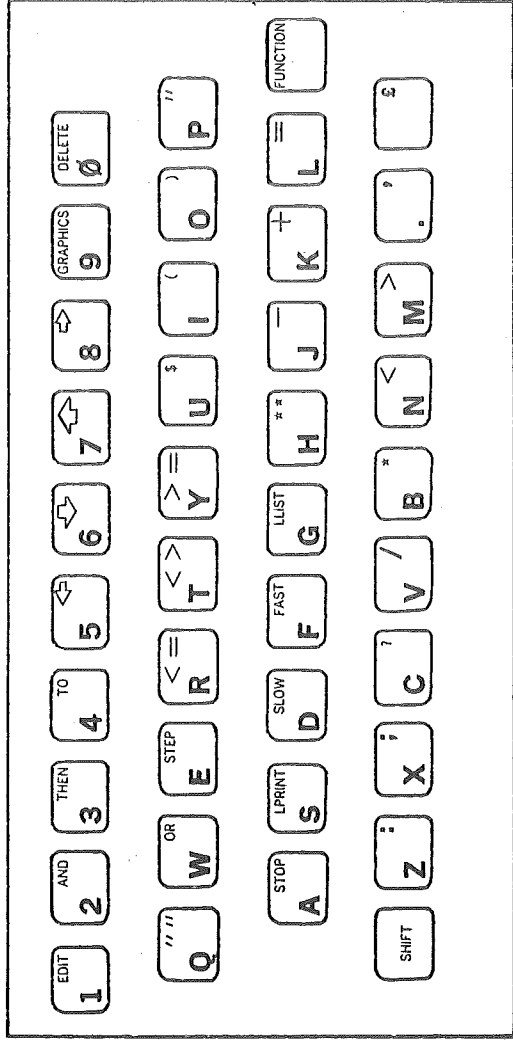
A tecla mais à direita, na terceira coluna de cima para baixo, chama-se **ENTER**.[3] O computador nada fará com a sua linha até você teclar **ENTER**. Tecle **ENTER**. A palavra ALO aparecerá no canto superior esquerdo do vídeo.

Você simplesmente forneceu ao seu computador o comando para imprimir a palavra ALO na tela da sua televisão. O computador sempre começará a imprimir do alto da tela. O que você estiver digitando sempre aparecerá na parte inferior do vídeo. Esta parte inferior agora, está mostrando o/o.* Isto é um *código de informação,* uma mensagem do computador dizendo que não houve nenhum problema na execução do comando. Um código de informação mostrado serve também como o cursor K .

Agora digite **PRINT** e **ENTER**. O ALO do alto da tela desapareceu. A entrada de um **PRINT** sozinho imprime uma linha em branco. A utilidade disto se tornará visível mais tarde.

Digite **PRINT** "ALO" novamente. Lembre-se de digitar P quando o cursor K estiver na tela e a seguir **SHIFT** P

---

[3] Em alguns computadores a palavra **ENTER** é substituída por **NEWLINE.**

Estes são os símbolos que operam quando a tecla **SHIFT** está pressionada.

para as aspas e as letras A, L, O e, depois, **SHIFT** P para as últimas aspas. Não tecle **ENTER** ainda.

As aspas na palavra ALO dizem ao computador que a palavra não deve ser interpretada como um comando. As aspas significam "Aceite tudo entre aspas como um todo e não examine o conteúdo" (isto é, não tente interpretar). Qualquer caractere pode ser colocado entre aspas. Um grupo de caracteres entre aspas chama-se *string*. Exemplo de strings:

"MEU NOME E JORGE"
"BOM DIA MEU CARO AMIGO"
"? * SS ((//?"
"123.456"
"1 + 1 = 10"

São todas *strings*, simplesmente porque estão entre aspas.

As teclas 5, 6, 7 e 8 na coluna mais alta do teclado do computador têm setas vermelhas impressas. Do mesmo modo como você vai proceder com tudo em vermelho no teclado, use aquelas setas, segurando o **SHIFT** e pressionando simultaneamente a tecla desejada. Mantenha **SHIFT** e pressione a tecla 5, a que tem a seta à esquerda. Observe que o cursor moveu-se um espaço para a esquerda. Ele estará à esquerda das aspas. Se você prestasse atenção, teria visto que na realidade as aspas moveram-se para a direita para dar espaço para o cursor. Tente agora pressionar o **SHIFT** e teclar 8. As aspas move-

ram-se um espaço para a esquerda e o cursor voltou para o fim da linha. Tecle **SHIFT** 5 novamente. O cursor moveu-se um espaço para a esquerda. Agora, pressionando **SHIFT**, tecle **DELETE.** A letra O desapareceu. **DELETE** sempre apaga um caractere (ou palavra-chave) imediatamente à esquerda do cursor. Ainda pressionando a tecla **SHIFT**, tecle 5 duas vezes. O cursor está agora entre as aspas e a letra A. A linha deve estar assim:

PRINT " K AL"

Solte o **SHIFT** e tecle L. Observe que o L foi para a posição do cursor, enquanto este andou um espaço para a direita. A linha deve estar assim:

PRINT "L K AL"

Digite um U e depois um Z. A tecla no canto inferior do teclado é chamada **SPACE.** A tecla próxima a esta tem impresso um ponto e uma vírgula em vermelho. Pressione o **SHIFT** e tecle a vírgula. Agora abandone o **SHIFT** e tecle o **SPACE.** Apareceu um espaço em branco. Digite um R, um A e um M. A linha ficará assim:

PRINT "LUZ, RAMAL"

Agora tecle **ENTER.** A linha desaparece e LUZ, RAMAL aparecerão no canto superior esquerdo da tela no lugar de ALO que tinha sido impresso antes. Observe que a posição do cursor na linha não faz diferença, uma vez que

ao pressionar **ENTER** o cursor nunca é impresso como parte de uma linha. Observe também que as aspas não estão impressas. Repare que a tecla **Q** tem duas aspas vermelhas impressas, as quais você tem que usar se quiser que fiquem dentro de uma string; elas são conseguidas com o **SHIFT** Q. Tente fazer isso:

PRINT "MARIA DISSE "OI" PARA MIM"

Alguma coisa curiosa aconteceu quando você digitou a linha, não? Quando você alcançou o M do MIM, você chegou ao fim da linha e, ao digitá-lo, a linha subiu uma posição e o resto continuou na linha seguinte. Uma linha na tela da TV contém apenas 32 caracteres, mas no computador as linhas podem ter qualquer tamanho. O computador começará uma nova linha, se você quiser. Tente agora teclar **ENTER** pois sem isso o computador não executa nada. A linha seguinte aparecerá no topo da tela:

MARIA DISSE "OI" PARA MIM

Observe que as duas aspas (**SHIFT** Q) foram reduzidas a aspas normais e são para uso exclusivo do computador a fim de evitar confusões sobre a posição final da string. Quando o computador executa o comando de imprimir a string, ele imprimirá apenas uma aspa.

A TV tem agora uma linha escrita no alto da tela. Vamos fazer uma "faxina na casa". Para apagar a linha, envie o comando que limpa a tela, **CLS**. **CLS** está escrito acima da tecla V. Uma vez que o cursor K esteja na tela, ao teclar o V, o **CLS** será impresso. Tente isto e depois tecle **ENTER**. A tela ficará branca.

Letras soltas não podem ser digitadas quando o cursor K estiver na tela. Em outras situações (as quais você verá no decorrer do livro) ser-lhe-á permitido digitar letras, mesmo se uma palavra-chave for necessária. Digitar letras separadamente (mesmo se elas *aparentarem* ser iguais a uma palavra) não é a mesma coisa que teclar uma palavra-chave. É importante lembrar isto. Se sua frase não for aceita pelo T/S 1000, procure verificar se você usou uma palavra-chave.

Palavras-chave são sempre digitadas com uma tecla e podem sempre ser apagadas teclando **DELETE (SHIFT** 0) uma vez. Neste livro, palavras-chave são escritas em maiúscula e em *negrito.*

Números, mas não letras, podem ser digitados no início de uma linha. Digite este número:

100

Observe que o cursor K é mostrado sempre. Digite a tecla P para fazer com que a palavra **PRINT** apareça. O cursor L é mostrado agora. Pressionando **SHIFT**, tecle P de novo, de modo que as aspas apareçam. Digite ALO, seguido de outras aspas. A linha deve ficar assim:

100 **PRINT** "ALO"

Agora tecle **ENTER.**

A palavra ALO não foi impressa na tela. Ao invés disso, a linha toda 100 **PRINT** "ALO" saiu da última linha para reaparecer no alto da tela.

Quando um número é digitado à frente de um comando, este número é chamado de *número da linha.* Coman-

dos com números de linha não são executados imediatamente. O computador copia a linha e guarda-a.

A tecla R é chamada **RUN**. Tecle R. A palavra **RUN** aparecerá na parte inferior da tela. Tecle **ENTER.**

A palavra ALO agora aparece na primeira linha da tela. Você acaba de escrever e "correr"[4] seu primeiro programa. Parabéns!

Comandos com números de linha chamam-se linha de *programa* ou *declarações.* Um grupo de linhas denomina-se *programa.* O computador não faz nada com linhas de programas até você mandá-lo "correr" as linhas.

Comandos e Programas

O T/S 1000 sozinho nada mais é do que uma caixa com circuitos eletrônicos embutidos. Para fazer o computador executar uma tarefa desejada, você tem que usar sua linguagem especial para formar os comandos que lhe ordenem o que fazer. Tente este comando novamente:

PRINT "ALO"

O computador imprime a palavra ALO na tela da TV.

Você só deu ao T/S 1000 um comando direto. A palavra **PRINT**, no comando acima, é um exemplo de uma palavra-chave. Uma palavra-chave diz ao computador que tipo de ação executar; **PRINT** diz ao computador para imprimir algo. O Timex/Sinclair 1000 tem 54 palavras-

---

[4]Alusão à palavra-chave **RUN** que em inglês quer dizer CORRER.

chave, e no mínimo uma deve ser usada toda vez que você enviar um comando para a máquina.

Antes que tratemos de comandos e programas, é importante entender como o computador mantém-se informado e atualizado com o seu programa: como é sua *memória*.

Muitas calculadoras têm memórias acessíveis à pessoa que trabalha com elas; quando você precisar usar um número muitas vezes, você o guarda na memória da calculadora. Algumas calculadoras mais caras têm várias memórias, cada uma podendo guardar um número.

Como você pode suspeitar, os computadores também têm memória para guardar valores para nova consulta no futuro.

As calculadoras têm um número fixo de posições de memória, mas você pode criar quantas você quiser no seu T/S 1000. Por exemplo, tente este comando:

```
LET NUM = 7
```

O comando **LET** cria uma *célula* (ou um espaço na memória) chamado **NUM** e nele guarda o valor 7. Conseqüentemente, o computador substitui o *valor*, neste espaço, pelo *nome* correspondente. Você pode usar um comando **PRINT** para descobrir o que está na célula **NUM**:

```
PRINT NUM
```

Um 7 aparece na tela.

Você pode usar um comando **LET** também para criar ou mudar o valor de **NUM**:

LET NUM = 117

Se você der um **PRINT**-NUM agora, o número 117 aparece na tela da TV. Como os valores destas células podem ser mudados a qualquer instante, eles são chamados *variáveis*. Ao contrário, números dados ao computador, por você (como o número 117 do lado direito do sinal de igual) são chamados de *constantes*. Um conjunto de caracteres entre aspas é chamado de *constantes strings* ou *literais*.

Uma variável (a célula) usada para guardar um número pode ter qualquer nome, contanto que comece por uma letra e contenha somente letras, números ou espaços. Todos os nomes a seguir são aceitos para uma variável:

X
BANANA
QUANT
ARI
ATE LOGO MAIS

O computador "ignora" espaços, de modo que o último nome da variável é para o computador igual a:

ATELOGOMAIS

É a mesma coisa que:

A T E   L O G O   M A I S

O sinal de igual (=) na declaração **LET** é usado para atribuir um novo valor à variável. Eles não têm o mesmo significado que o sinal de "igual" que você está acostumado a usar em aritmética. Quando usado com o **LET**, o sinal de "igual" significa "tem o valor" ou "é igual ao valor" de modo que:

LET PATRICIA = 20185

Pode ser lido como "Guarde a variável PATRICIA com o valor 20185. Em outras palavras, o valor 20185 é *atribuído* à variável PATRICIA. Assim, qualquer declaração que comece com a palavra-chave **LET** é chamada declaração de atribuição.
O comando **LET** pode também ser usado assim:

LET NUM = NUM + 10

Entre com este comando e depois com **PRINT**-NUM. NUM agora está com o valor 127. O computador primeiramente acha o valor da expressão ao lado direito do sinal de "igual" (neste caso NUM + 10) e então coloca (atribui) o valor na variável do lado esquerdo. O valor atual de NUM é usado para calcular o novo valor da expressão. NUM + 10 significa "Some 10 ao valor atual de NUM e coloque o resultado em NUM."

Podem-se usar variáveis também para formar strings, que no caso são chamadas de *variáveis strings.* Strings são simplesmente grupos de caracteres. Não têm valor numérico. Você não pode, por exemplo, multiplicar duas

strings. As variáveis strings têm seus nomes formados por uma letra seguida de um cifrão ($). Alguns nomes de variáveis, válidos, são:

A$, Z$, I$, P$

Como existem 26 letras no alfabeto, existem 26 variáveis strings válidas.

Criar uma variável string dando-lhe um valor, é igual a criar uma variável numérica. Tente digitar este comando:

```
LET A$ = "MARIA FOI A PADARIA"
```

Agora digite este comando:

```
PRINT A$
```

O computador imprime a string.

Agora, tendo em mente o conceito de memória e variáveis, podemos verificar como o computador "lembra-se" dos comandos. Quando uma lista de comandos diretos é dada, cada comando é executado como você o digitou.

Digite a seguinte seqüência de comandos:

```
LET A = 10
LET B = 5
LET C = 100
PRINT A + B + C
```

Estes comandos são equivalentes a:

```
PRINT 10 + 5 + 100
```

Entretanto, usando variáveis, você pode escrever programas muito mais flexíveis, eficientes e úteis, como você verá adiante.

Você pode fazer uma lista de comandos, dando a cada um deles um número que os identifique, chamado *número de linha,* e então dizer ao computador que faça tudo que está escrito na lista.

Uma lista de comandos numerados chama-se programa e os comandos em um programa são chamados declaração. Digite esta declaração:

10 **PRINT** 10 + 5 + 100

Quando você tecla **ENTER**, a declaração sai da última linha da TV e reaparece ao alto. Você agora tem um programa de uma linha na memória de seu computador. O computador não faz coisa alguma com seu programa até você dizer a ele para executar o programa teclando:

RUN

O computador imprime 115. Você pode "correr" o seu programa de uma linha quantas vezes você quiser, e toda vez o computador imprimirá 115.

Tente digitar este programa:

```
10 LET A = 10
20 LET B = 5
30 LET C = 100
40 PRINT A+B+C
```

Digitando uma nova declaração de número 10, apaga-se a antiga. Agora digite **RUN**. Mais uma vez o computador imprimirá 115 na tela.

Aqui está, em detalhes, o que aconteceu com o nosso programa. O computador pegou o primeiro valor 10 e o guardou na variável A. Depois, guardou o segundo 5 na variável B e o terceiro valor na variável C. A última linha, a 40, diz ao computador que ele deve somar A, B, C e imprimir o resultado. O computador, conseqüentemente, lê o valor contido em cada variável, e então soma todas elas, imprimindo o total na tela da TV.

Troque as linhas 10, 20, 30 digitando:

```
10 INPUT A
20 INPUT B
30 INPUT C
```

À medida que você entra[5] com as declarações, o computador acrescenta-as à lista do programa no alto da tela. A declaração 40 **PRINT**-A + B + C está ainda na memória; assim, você não precisa digitá-la de novo. Tecle **RUN** para executar o programa.

Nada aconteceu, aparentemente. A declaração **INPUT** A significa "Retire o valor de A pelo teclado" (caso a variável A não exista, o computador cria uma com este nome).

---

[5] Alusão à palavra-chave **ENTER**, que em inglês quer dizer ENTRAR.

Para dar seguimento ao exemplo, digite 50 e **ENTER.** O computador executa a declaração e espera pelo valor de B. Entre com 20. Agora ele espera por um valor para C. Entre com 7. Finalmente, após o último valor ter sido digitado, o computador imprime 77, a soma A + B + C.

Este é um exemplo comum de por que um programa é superior a uma simples lista de comandos. Este programa simplesmente somou uma lista pequena de números; se a lista fosse longa e complexa, o computador iria economizar um tempo enorme.

Quando você digita um comando direto (um comando sem um número de linha), o comando é executado imediatamente, e a linha do comando desaparece. Conseqüentemente, comandos diretos são executados em *modo imediato,* também chamado de *execução.*

Quando você associa números de linha a comandos, o computador guarda as linhas e não as executa até que você ordene. Linhas de programa são executadas no *modo de programação.* Com poucas exceções, os comandos podem ser usados dos dois modos. No modo de programação, entretanto, você pode parar sua execução, dar um **RUN**, editá-lo etc. Geralmente, um computador no modo imediato é uma calculadora poderosa e de grande maleabilidade. Já o modo de programação permite que o computador funcione como um computador de verdade.

Você já aprendeu a instalar o seu Timex/Sinclair 1000, a operar o seu teclado e a interpretar o que você vê na TV. Você recebeu o conceito de strings, variáveis e coman-

dos. E agora sabe como colocar o seu T/S 1000 para trabalhar. Nos dois próximos capítulos, você vai escrever seu primeiro programa que funciona de verdade: um jogo de dados. Você também aprenderá mais técnicas novas de programação para adicionar à sua ferramenta programável.

3

CONCEITOS INTRODUTÓRIOS

Você deve ter notado que a parte inferior de sua tela freqüentemente mostra dois números como 0/100[6]. Os números que aparecem aqui são *códigos de informação* e servem para informá-lo que o computador interrompeu a execução do seu programa. O 0 significa que o computador executou com sucesso o que você lhe pediu, enquanto que o 100 significa que ele ficou sem linhas para executar, após a linha 100. Digite **PRINT** e **ENTER** para apagar a palavra ALO.

---

[6] Para observar este fato e outros explicados neste capítulo, repita as operações apresentadas na página 15-16.

Além do **RUN**, o computador pode ser instruído para executar o programa com um comando **GOTO**. A tecla G tem impressa, acima dela, a palavra **GOTO**. Agora que o cursor **K** é mostrado de novo, tecle G. **GOTO** aparece na tela. Digite 100. A linha deve ficar assim:

GOTO-100

O comando significa "Comece a execução a partir da linha 100". Tecle **ENTER**.

A palavra ALO aparece novamente. Usando **RUN** ou **GOTO** um programa pode ser executado quantas vezes se quiser. **RUN** e **GOTO**, entretanto, têm uma pequena diferença, que se tornará clara mais adiante. Por enquanto, digite a seguinte linha:

200-**GOTO**-100

Tecle **ENTER**. Esta linha e a linha 100 que você já havia digitado anteriormente aparecem no alto da tela. Novas linhas numeradas são adicionadas, em ordem crescente à listagem do programa, automaticamente. Sempre que você quiser, será possível começar a executar a listagem: digite **RUN** e **ENTER**.

A parte esquerda da tela ficará repleta de ALOs.

Quando o computador executa o comando da linha 100, ALO é impresso na tela. O computador, então, executa a linha 200 que diz a ele para pular para a linha 100 e executá-la. Este processo continua até que a tela fique cheia. A linha inferior da tela mostra 5/100. O 5 é um có-

digo de erro que informa que a tela ficou cheia e o 100 significa que o computador estava tentando executar a linha 100 quando a tela ficou repleta. Lembre-se de que um código de informação é um sinal de que o seu programa parou de ser executado, geralmente devido a um erro. O T/S 1000 tem 15 códigos de erro no total; eles estão listados no Apêndice F no final deste livro.

A tecla C é chamada de **CONT** (de **CONTINUE**). Tecle C e a palavra **CONT** aparece na tela. Tecle **ENTER**. Outra tela cheia de ALOs aparece e a mensagem 5/100 é impressa novamente. Tecle **CONT** e **ENTER**, novamente. A tela fica repleta mais uma vez. Você poderia ficar fazendo isso indefinidamente. O programa de duas linhas:

```
100 PRINT "ALO"
200 GOTO 100
```

ficará dando *loops*[7] para sempre. Este tipo de loop chama-se *loop infinito* porque ele, por si só, nunca terminará. Na verdade, este loop infinito pára quando a tela fica cheia, mas somente porque não tem mais linhas, e não porque o loop foi "quebrado". Para terminar o loop sem desligar o computador, use a palavra-chave **BREAK**, localizada na tecla **SPACE**. O **BREAK** é a sua maneira de interromper um programa enquanto está sendo executado.

---

[7] Loop: volta. Dá idéia de círculo, isto é, um círculo nunca termina. É uma expressão muito usada em programação BASIC.

Para ver o seu programa, tecle a palavra-chave **LIST**. Como **LIST** está localizado acima da tecla K, tecle K para fazer com que a palavra **LIST** apareça. Tecle **ENTER**. O computador imprime a sua listagem de comandos — o programa de duas linhas será impresso no alto da tela. **LIST** imprime o seu programa na tela.

Digite a seguinte linha:

50 **PRINT** "UMA TELA DE ALOS"

Tecle **ENTER**.

A linha 50 aparece acima das linhas 100 e 200.

Quando se "entra" com uma linha de programa, o computador sempre a coloca em ordem numérica. Este é um processo significativo porque programas são executados do menor para o maior número de linha (exceto quando você insere uma declaração **GOTO** para mudar o fluxo lógico do programa).

Observe que existe uma pequena seta dentro de um quadrado negro entre o 50 e o **PRINT**. Esta seta aponta a *linha corrente* do programa. Olhe as teclas 5, 6, 7 e 8. Cada uma delas tem uma seta impressa. Você já usou as setas que apontam para a direita e para a esquerda para movimentar uma *linha na parte inferior da tela*. Entretanto, as setas que apontam para cima e para baixo são usadas para movimentar o *cursor da linha corrente*. Mantenha a tecla **SHIFT** pressionada e tecle 6. O cursor de linha corrente movimenta-se para baixo, para a posição da linha 100.

A palavra em vermelho na tecla 1, no canto superior esquerdo do teclado é o **EDIT**. Quando você conserta uma frase num pedaço de papel, você usa um lápis e uma borracha para fazer correções. Igualmente, o computador tem comandos para inserir ou retirar partes de uma linha de programa. O processo de fazer mudanças em seu programa chama-se *editar*. Pressione o **SHIFT** e tecle **EDIT**.

A linha 100, **PRINT** "ALO" é copiada na parte inferior da tela. Você agora pode fazer mudanças nesta linha. Use a seta que aponta para a direita (**SHIFT 8**) para mover o cursor até o final da linha. Acrescente um ponto-e-vírgula (;) na linha pressionando **SHIFT** e teclando X. A linha agora fica assim:

100 **PRINT** "ALO";

Tecle **ENTER** e a nova linha reaparece acima da original. Agora tecle **RUN** e **ENTER**. A tela ficará repleta de ALOs de um canto ao outro.

O ponto-e-vírgula no final de uma linha **PRINT** significa "Não comece a imprimir em outra linha — continue na mesma posição". Observe que no fim de cada linha, o ALO está dividido. Quando uma linha está completa, o computador a abandona e, automaticamente, começa a escrever no início da próxima. Uma linha contém 32 caracteres; 22 linhas preenchem a tela e mais a 23ª e a 24ª linha no inferior da tela que são usadas para imprimir códigos de informação e linhas de comandos.

Digite a seguinte linha (não coloque o 5 entre aspas):

PRINT 5

Tecle **ENTER**. Um 5. aparece no canto superior esquerdo da tela. Os números não precisam estar entre aspas quando você os imprime. Na verdade, o computador os trata diferentemente quando eles estão entre aspas (isto é, como qualquer outra string). Tente digitar as duas linhas seguintes, teclando **ENTER** após cada uma (daqui em diante, você não será mais avisado que tem de digitar **ENTER** depois de cada linha; mas se você não o teclar, o computador não fará nada). Pense no **ENTER** como um ponto final do seu comando.

PRINT "1 + 3 + 4 + 5"
PRINT 1 + 3 + 4 + 5

Com a primeira linha, o computador só imprime a *string* 1+ 3 + 4 + 5 na tela da TV. Com a segunda, ele imprime o número 13 na TV, desde que 13 seja a soma dos números. Quando você manda uma string para o computador, ele a trata como uma unidade simples. Quando você dá ao computador um número, ele usa o valor deste número. Logo, quando você "entra" com a linha **PRINT** 1 + 3 + 4 + 5, o computador soma todos os números e os entende como um valor.

Digite um **PRINT** e uma aspas novamente (**PRINT** ") (não se esqueça do **ENTER!**).

A linha fica na mesma posição. Observe que próximo ao cursor há um quadrado negro com um **S** dentro dele. Ele indica que o computador não podia aceitar a linha que foi escrita erroneamente. Da mesma forma que você pode ter problemas ao entender uma frase gramatical-

mente incorreta, o computador não pode entender um comando incorreto. Ele só entende um vocabulário extremamente restrito. Se você não "pronunciar" corretamente uma palavra ou entrar com um comando (ou um grupo de comandos) incorretamente, o computador não saberá o que você quer fazer. Ele rejeitará a linha e colocará um S antes da parte que ele não entendeu.

Em nosso exemplo acima, o computador ficou confuso porque você deu apenas uma aspas. As aspas *devem* vir sempre aos pares. Erros deste tipo são indicados por um S (que vem de *sintaxe*).

Por um problema de abreviação, usaremos o termo **SHIFT**/nome da tecla para significar "pressione **SHIFT** e digite a tecla com o nome". Por exemplo **SHIFT**/A significa "tecle **SHIFT** e a tecla A simultaneamente".

Expressões

Os exemplos q você tem visto até agora incluem algumas operações matemáticas. Especificamente, você viu esta linha:

```
PRINT 1+ 3 + 4 - 5
```

Um grupo de números ligados por sinais de adição (+) ou outros operadores chama-se *expressão numérica.* Você provavelmente está acostumado com operações matemáticas comuns: adição, subtração, multiplicação e divisão. Quando você usa uma destas operações, está

criando uma expressão numérica. O T/S 1000 contém um *manipulador de expressões* que é usado toda vez que uma expressão é encontrada em um comando. O manipulador de expressões calcula o valor final da expressão. Já que este manipulador é chamado quando uma expressão é achada, você pode usar uma expressão numérica em quase todo lugar que um valor numérico for usado. A única exceção é a numeração de linhas de programas. O seguinte comando não pode ser aceito:

5 + 100 **PRINT** "ALO"

Você pode, entretanto, usar uma expressão no lugar onde um número de linha é necessário e usar uma declaração **GOTO** assim:

200 **GOTO** 100 + BASE

O manipulador de expressões irá, primeiramente, achar o valor contido na variável BASE e então soma 100; o programa irá pular para a linha que contém o valor final da expressão. É lógico que a variável BASE tem que ser criada antes que o programa execute a linha 200.

Os operadores que você pode usar são:

adição +
subtração −
multiplicação *
exponenciação **
divisão /

Estes operadores aparecem em vermelho nas teclas K, J, B, H e V. Digite-os pressionando **SHIFT** e as teclas correspondentes.

A adição (+) e subtração (−) têm seu formato usual.

A multiplicação usa como símbolo o asterisco (*) para evitar confusão com símbolos mais comuns da multiplicação (X, .), com a letra X e com o ponto.

Na divisão (/), o primeiro número é dividido pelo segundo. Tente este exemplo:

```
PRINT 10/2
```

O computador divide 10 por 2 e imprime a resposta: 5.

Frações comuns são escritas na forma decimal. Por exemplo, 10 1/2 é escrito como 10.5. Você poderia escrever 10 + 1/2, mas a forma decimal é mais simples.

Nosso último operador, a exponenciação, pode não ser muito familiar. A exponenciação é considerada como "elevar um número a uma potência". É usualmente escrito com a potência como subscrita e com o dígito da potência meia linha acima do número da base. Por exemplo, 2 elevado à quinta potência é escrito como: 2^5. Isto significa que o 2 é multiplicado por ele mesmo 5 vezes (2 * 2 * 2 * 2 * 2) e o resultado é 32. Aqui está um exemplo que mostra a utilidade de exponenciação.

Existiam aproximadamente 100 Pilgrims no *Mayflower.*[8] O crescimento normal de uma população é de

---

[8] Os Pilgrims foram os primeiros colonos puritanos que chegaram no navio *Mayflower* aos EUA e fundaram, em 1620, a colônia de Plymouth.

aproximadamente 2% ao ano, e a população de descendentes do *Mayflower* dobra a cada 35 anos. Isto significa que a população original de 100 ou mais Pilgrims dobrou aproximadamente 10,3 vezes nos últimos 360 anos.

Se você digitar isto numa declaração, você verá quantos descendentes do *Mayflower* podem estar nos EUA. Digitando dois asteriscos normais (*) juntos, o comando não vai funcionar. Você deve usar o asterisco duplo (**) na tecla H (**SHIFT**/H).

```
PRINT 100 * 2 ** 10.3
```

As cinco operações válidas que você pode usar em expressões numéricas não são executadas necessariamente da esquerda para a direita. Tente digitar este comando:

```
PRINT 5 -4*3
```

O computador imprime — 7 (o oposto de 7). Se o calculador de expressão fosse simplesmente executá-la da esquerda para a direita, ele iria, primeiramente, subtrair 4 de 5, depois multiplicaria 1 por 3 e o resultado seria 3. Na verdade o calculador de expressões executa a operação 4*3 em primeiro lugar e depois subtrai o resultado, 12, de 5.

O calculador de expressão executa, primeiramente, funções e exponenciação. Após isso, ele faz as multiplicações e divisões seguidas das adições e subtrações.

Se existir mais de uma multiplicação ou divisão, ele as fará da esquerda para a direita. Se existir mais de uma

adição ou subtração, elas também serão executadas da esquerda para a direita.

Qualquer coisa entre parênteses é calculada primeiramente, usando as prioridades já mencionadas. Parênteses podem ser utilizados a qualquer hora para fazer com que suas expressões possam ser mais fáceis de ser visualizadas e entendidas. Qualquer coisa precedida por um sinal de menos (que parece um sinal de subtração mas indica um número que é menor que zero) é calculada imediatamente após a exponenciação.

Resumindo, a ordem de prioridade é:

1. conteúdo dos parênteses
2. funções
3. exponenciação (**)
4. números negativos (–)
5. multiplicação e divisão (*,/)
6. adição e subtração (+, –)

Strings, entretanto, têm somente um operador: +. O operador + concatena duas strings. Tente este comando:

```
PRINT "RATO" + "GATO"
```

O computador imprime RATOGATO. Qualquer espaço fora das aspas é ignorado.

Variáveis strings podem ser usadas como expressões strings. Digite este comando:

```
LET A$ = "MULA" + "VEADO"
LET = B$ = "CAVALO"
PRINT A$ + B$ + "PEIXE"
```

O computador imprime MULAVEADOCAVALOPEIXE na tela da TV.

Uma expressão string pode ser sempre usada no lugar de uma constante string.

Existem 4 comandos que afetam todas as variáveis do computador: **CLEAR**, **NEW**, **RUN** e **LOAD**.

CLEAR apaga todas as variáveis.

RUN destrói todas as variáveis e depois executa qualquer programa gravado no computador.

NEW limpa toda a memória: todas as variáveis e programas são perdidos para sempre.

O comando **LOAD** destrói todas as variáveis e programas quando ele lê outros programas do cassete. As variáveis guardadas no cassete são preservadas.

Todos os quatro comandos limpam a tela. Você pode limpar a tela sem mudar variáveis ou programas, usando o comando **CLS**.

À medida que sua capacidade para programar for aumentando, você descobrirá que os programas se tornarão mais complicados. Quando isto ocorrer, você, progressivamente, sentirá mais dificuldade em se lembrar o que determinadas linhas pretendem fazer em seu programa. Os programas em que você trabalhou por muitas horas lhe parecerão totalmente diferentes em poucas semanas, e você será forçado a reconstruir o pensamento

que o conduziu a escrever as linhas. Geralmente, é mais difícil redescobrir um pensamento do que tê-lo pela primeira vez! Não seria bom se nós pudéssemos deixar anotações para nós mesmos, explicando o que pretendíamos, de modo que após algum tempo nós (ou qualquer outra pessoa) pudéssemos ter nossos critérios dentro do nosso programa genial?

Felizmente, o BASIC nos deu mais uma ferramenta: a declaração **REM** (do inglês: **REM**ARK = COMENTÁRIO). Quando seu computador encontra uma linha como:

100 **REM** A$ = O NOME DO JOGADOR

ele simplesmente ignora tudo que segue o **REM**. Nenhum código de erro será gerado porque o **REM** assinala ao programa que pule para a próxima linha.

Da próxima vez que você quiser ver o seu programa, você se lembrará que A$ =O NOME DO JOGADOR e não terá que calcular a variável pelo contexto do programa.

Infelizmente o espaço de memória é tão precioso no basic do Timex/Sinclair 1000 que nós preferimos não incluir declarações **REM** nos programas apresentados neste livro. Mas o bom uso de declarações **REM** é geralmente considerado tão essencial para a prática da boa programação, quanto o pensamento lógico. Se você estiver pretendendo adquirir, no futuro, uma extensão de memória de 16K, nós recomendamos, sinceramente, que você comece a guardar para você mesmo, estas anotações importantes.

4

PROGRAMA DE JOGO DE DADOS

Um programa muito simples é um jogo de dados. Ele é de uso muito limitado: seu microcomputador nunca será tão portátil ou tão rapidamente utilizável quanto um par de dados. Por outro lado, um programa de dados tem a vantagem de usar os tipos de declaração de programação mais comuns; conseqüentemente, uma vez que você entenda como um programa de dados funciona, você estará capacitado a escrever seu próprio programa.

Ao receber um problema de programação, o primeiro passo a ser dado é fazer um modelo exato daquilo que você deseja que o programa execute.

Você poderia produzir sua própria lista, mas esta foi usada para delinear o programa descrito neste capítulo.

1. Descobrir quantos dados são usados pelo jogador.

2. Dar ao jogador a oportunidade de terminar o jogo.

3. Escolher o lado do dado.

4. Ir ao item 2 quantas vezes for necessário.

Aqui está o programa todo. Digite-o agora e o execute para sentir como ele funciona. Após isto, neste capítulo, o programa será examinado em detalhes. Lembre-se de que as palavras em negrito são as palavras-chave, que devem ser inseridas com o toque de uma única tecla. **INPUT**, **IF**, **CLS**, **FOR**, **PRINT**, **NEXT** e **GOTO** (a primeira palavra-chave em cada linha de programa) serão impressas quando você pressionar a tecla que está abaixo das palavras-chave.

Você obtém as palavras-chave **THEN**, **STOP** e **TO**, que são escritas em vermelho, pressionando **SHIFT** e as teclas nas quais elas estão escritas. As palavras-chave **INT** e **RND** são de novo tipo; são funções que serão explicadas mais tarde. Por agora, quando tiver que teclar **INT**, mantenha o **SHIFT** pressionado e tecle **ENTER** (que tem FUNCTION escrita em vermelho). O cursor mudará de **L** para **F** . Depois pressione a tecla com o **INT** escrito *abaixo* dela e a palavra-chave **INT** aparece. O cursor **F** também mudou para um **L** . Para digitar **RND**, faça a mesma coisa: digite **SHIFT/ENTER** e então pressione a tecla R, na qual o **RND** está escrito a seguir.

```
 50 PRINT "QUANTOS DADOS?"
100 INPUT A
200 CLS
300 IF A = 0 THEN STOP
500 PRINT INT (RND*6+1)
600 LET A = A-1
700 GOTO 300
```

Agora, rode o programa. Digite-o:

RUN

Não se esqueça de teclar **ENTER**. Quando você tiver acabado de jogar o programa, termine-o entrando com 0 para número de dados a rolar.

O Passo 1 — no qual achou-se quantos dados se quis jogar — foi fácil traduzir para o programa. Você já tinha visto, antes, a declaração **INPUT** e, aqui, uma é necessária.

```
100 INPUT A
```

A declaração 200 contém o **CLS** (limpa a tela).[9] Após você entrar com o número de dados a serem mostrados, o programa apaga tudo da tela para deixar um campo branco para a nova jogada. Você também precisa encontrar um modo de terminar o programa. Se a pessoa que o está usando pedir ao **INPUT** 0 dados, o programa detecta

[9]**CLS** é abreviação de CLEAR SCREEN que quer dizer LIMPAR A TELA.

o 0 e pára. É necessário uma declaração que diga "Se A for igual a 0, pare o programa". Isto se faz com a declaração:

300-IF-A= 0-THEN-STOP

Este é um exemplo da declaração **IF ... THEN.**

A declaração **IF ... THEN** permite ao computador tomar decisões. Geralmente uma declaração **IF... THEN** tem a seguinte forma:

(número da linha **IF** (condição for verdadeira) **THEN** (comando)

Você pode ter notado que o cursor K apareceu após você digitar a palavra-chave **THEN.** O comando seguinte ao **THEN** pode ser qualquer comando BASIC legal. No caso da declaração 300, o comando é **STOP,** que significa "Pare o programa".

O **IF** é conhecido como *expressão relacional.* Toda expressão relacional ou é verdadeira ou falsa. Por exemplo: a declaração 300 em expressão **IF**-A = 0. Se A for realmente igual a 0, o comando que vem depois do **THEN** é executado. Se A não for igual a 0 então o **STOP** não é executado e o programa pula imediatamente para a próxima linha.

A relação usada nesta expressão relacional particular é "igual", mas igualdade não é a única expressão relacional válida. Existem seis modos pelos quais dois valores podem ser comparados. O primeiro valor pode ser:

1. maior que (>),
2. menor que (<),
3. igual a (=)
4. menor ou igual a (<=),
5. maior ou igual a (>=),
6. diferente de (<>)

} O segundo valor

Em cada caso, se a expressão relacional for verdadeira, o comando após o **THEN** é executado. Se a relação for falsa o comando imediatamente após o **THEN** é ignorado e o programa continua na execução da linha seguinte.

Aqui estão alguns comandos que você pode testar usando o **IF ... THEN** (digitar comandos sem números de linha não afeta as declarações do programa que você já tinha digitado).

```
LET CAO = 7
LET GATO = 8
IF CAO < GATO THEN PRINT "GATOS SAO MELHO-
RES"
```

A mensagem é impressa na TV desde que o valor de CAO, 7, seja menor que o valor de GATO, 8.

O comando depois do **THEN** pode ser outra declaração **IF ... THEN**. Tente isto:

```
IF CAO<>GATO THEN IF GATO = 8 THEN IF CAO = 7
THEN PRINT "GATOS SAO 8"
```

Desde que a expressão relacional em cada uma das três declarações **IF ... THEN** seja verdadeira, a mensagem é impressa.

Existe uma maneira mais fácil de escrever declarações compostas de **IF ... THEN** muito longas. Você pode juntar expressões relacionais com duas palavras-chave **AND** e **OR** (digite estas palavras-chave pressionando **SHIFT** e 2 para obter **AND** e **SHIFT**/W para obter **OR**). Cada um deles pode ser usado para conectar duas expressões relacionais.

Tente digitar esta declaração:

```
IF CAO<>GATO AND GATO = 8 AND CAO = 7 THEN
PRINT "GATOS AINDA SAO OITO"
```

Esta declaração tem exatamente o mesmo efeito dos **IFs** compostos que você digitara antes, mas este é mais eficiente (economiza memória e tempo de execução) porque o computador tem menos palavras-chave para interpretar e executar.

Se duas expressões relacionais são conectadas com o **AND**, as duas juntas somente são verdadeiras se cada uma individualmente for verdadeira. A declaração **IF** acima pode ser traduzida para: Se CÃO não é igual a GATO e se GATO for igual a 8 e CÃO for igual a 7 então imprima "GATOS AINDA SÃO OITO."

Quando duas expressões relacionais são conectadas com **OR**, as duas juntas serão verdadeiras se pelo menos uma das duas for verdadeira. Tente isto:

IF CAO > GATO **OR** CAO < GATO **THEN PRINT** "CACHORROS SAO DIFERENTES"

A mensagem é impressa porque, mesmo que a primeira expressão relacional (CAO>GATO) seja falsa, a segunda (CAO <GATO) é verdadeira.

Você pode usar **AND** e **OR** para resumir quantas expressões relacionais você quiser.

Voltando ao programa de dados. O Passo 2 do plano deste programa era pegar um lado do dado. O próprio programa deve pegar um número representado pelo número de pontos de um dos lados do dado.

Quando um dado é rolado, ele pára com um dos lados para cima; qual o lado que cairá com a face para cima é puramente uma questão de sorte. Em termos computacionais, pode-se dizer que jogar o dado é pegar um número randômico (aleatório) entre 1 e 6. *Randômico* indica que um número tem tanta chance quanto qualquer outro, cada vez que o dado é jogado. Suponha que o resultado tenha sido 6. Se você pegar o dado e jogá-lo de novo, você tem um sexto de chance de que o resultado seja 6, 3, 1, 2, 4 ou 5. Cada vez que você rolar o dado, existe uma chance igual de cada um dos lados cair com a face para cima.

O T/S 1000 tem capacidade de gerar números aleatórios através do uso da função **RND**.

Funções são programas especiais guardados permanentemente no computador. Cada função produz algum número ou letra.

As funções podem ser usadas no lugar de uma constante ou em uma expressão. Serão tratadas com mais profundidade no próximo capítulo, mas por agora só as veja como programas permanentes preparados e que são parte das ferramentas do BASIC.

As funções são sempre indicadas por palavras-chave. Para digitar uma palavra-chave de uma função, mantenha o **SHIFT** pressionado e tecle **ENTER**, primeiramente. Na tecla **ENTER** você verá a palavra FUNCTION em vermelho. Pressionando **SHIFT/ENTER**, você está invocando as facilidades das funções. Observe que nada foi impresso mas o cursor mudou agora para um cursor F. Isto indica que o computador espera pela próxima entrada que será uma função.

Observe que a maioria das teclas tem nomes logo abaixo delas. Por exemplo Q, W e E têm **SIN**, **COS** e **TAN**. Estas são palavras-chave de função, e quando o cursor F é mostrado e você tecla alguma delas, a palavra-chave da função é impressa. (Na verdade, nem todas estas palavras abaixo das teclas são funções. Algumas servem para outros propósitos mas são escritas abaixo das teclas por conveniência. Em qualquer caso, entretanto, você pode digitar as palavras-chave abaixo das teclas usando o cursor F). **RND** está abaixo da tecla T; então, para usar a função **RND**, tecle **SHIFT/ENTER**, e depois esqueça as duas teclas e tecle T. O cursor F permanece somente por um toque e depois é imediatamente trocado pelo cursor L .

RND é uma função fácil de ser usada. Tente digitar este comando:

PRINT RND

O computador imprime um longo número entre 0 e 1. Tente novamente **PRINT RND** por muitas vezes; você deve obter um número diferente a cada tentativa. Os números que vão sendo gerados não são realmente randômicos, mas são "pseudo-randômicos".

A seqüência destes números foi programada para prover a aparência de aleatoriedade, de modo que quando você entra com um programa que requer a geração de um número aleatório (tal como o nosso programa de dados), a função **RND** possa efetivamente satisfazer a esta necessidade. Daqui em diante, quando nos referirmos à geração de números randômicos pelo computador, você saberá que estamos falando de números pseudo-randômicos.

O computador sempre produz um número aleatório entre 0 e 1; mas um dado sempre produz um entre 1 e 6. Tente digitar o seguinte comando:

PRINT 1+6* RND

Este comando irá sempre produzir um número entre 1 e 6. Entretanto, ele não é exatamente como seu dado porque um dado sempre produz um número *completo* ou *inteiro*. Isto é, um dado sempre dá: 1, 2, 3, 4, 5 ou 6 mas nunca, por exemplo, 1,5. A primeira vez que você digitar

PRINT 1 + **RND** * 6 o computador deve dar a você, por exemplo, 5,7415161. A segunda vez ele pode dar 2,6150818.

Para fazer um computador agir como um dado, necessitaríamos eliminar tudo à direita do ponto decimal. A função **INT** do T/S 1000 faz exatamente isto: converte um número em um inteiro retirando tudo à direita do ponto decimal. Tente digitar o seguinte comando (lembre-se de que para obter o **INT** você deve pressionar **SHIFT/ENTER** de modo que o cursor **F** apareça e, então, pressionar a tecla R. Como o cursor **F** permanece com apenas um toque, você terá que obtê-lo novamente para digitar **RND**):

```
PRINT INT (+ 6 * RND)
```

O computador imprime um número inteiro entre 1 e 6.

Os parênteses na declaração fazem com que o **INT** retorne ao número inteiro de toda a expressão dentro dos parênteses. As funções geralmente operam somente com o digito imediatamente seguinte à palavra-chave da função. Sem parênteses, o **INT** iria retornar ao inteiro de 1, o que não iria ser de muita ajuda.

Como você pode ver, as funções podem ser muito úteis na edição de um programa. Existem 21 funções no T/S 1000; as mais usadas do Capítulo 8 até ao 11, em programas, são as mais importantes. Dê uma olhada no manual que acompanha o seu computador para uma descrição do resto das funções, quando você precisar delas.

A forma final do nosso comando que imprime o número inteiro randômico é:

```
500 PRINT INT (1 + 6 * RND)
```

Estão completos os passos 1 e 2 de nosso plano de programa. O passo 3 é voltar para o 2 para ver quantos números são necessários.

A declaração 100, INPUT-A, obtém do jogador o número de dados a serem jogados. O programa deve, então, executar a declaração 500 o número correto de vezes (isto é, o número de vezes é igual ao valor de A).

Aqui está o programa de novo para refrescar sua memória:

```
 50 PRINT "QUANTOS DADOS?"
100 INPUT A
200 CLS
300 IF A = 0 THEN STOP
500 PRINT INT (1 + 6 * RND)
600 LET A = A - 1
700 GOTO 300
```

Escrevendo o programa usando LET-A = A- 1 e GOTO-300 toda vez que um número for impresso, 1 é subtraído de A. Quando A é igual a 0, a frase condicional na declaração IF na linha 300 será verdadeira e o programa irá parar.

Isto completa nosso plano inicial para o programa do jogo de dados. Você já percorreu um longo caminho e aprendeu muitos conceitos novos. Se nada ficou claro para você até este ponto, reveja tudo, até que você se sinta seguro; então, leia o próximo capítulo que grava o seu programa de dados no cassete para um uso posterior.

Nós não terminamos nosso programa de dados; entretanto, no Capítulo 7 adicionaremos novas "incrementações" para que os dados apareçam realmente na tela! Se você estiver realmente ansioso para continuar com mais programação, pule direto para o Capítulo 6.

5

GRAVANDO PROGRAMAS EM FITA

Ao contrário de muitas pessoas, os computadores não se importam em repetir indefinidamente tarefas monótonas. Se fosse necessário digitar um programa toda vez que você quisesse jogar algo, fazer um balanço do seu talão de cheques ou simplesmente editar um programa, você logo ficaria chateado e frustrado. Com a ajuda de um gravador cassete comum, o seu computador é capaz de salvá-lo do tédio.

Como Gravar em Cassete

Depois de trabalhar exaustivamente digitando um programa, é uma pena que todo seu trabalho desapareça

quando o T/S 1000 for desligado. Felizmente, com um gravador cassete barato, você poderá guardar facilmente seus programas em fitas cassetes. Poderá também passar programas de outros cassetes para o seu computador. Nos próximos parágrafos discutiremos os tipos de gravadores e fitas que darão o melhor resultado.

O Gravador Cassete

O melhor tipo de gravador cassete é aquele que é barato e monaural (*não* é estéreo). Um tape-deck que é parte de um sistema estéreo grande, definitivamente não é recomendado.

O gravador deve ter um microfone externo **(MIC)** e uma saída áudio **(EAR).** Junto com o seu computador Timex existe um fio curto duplo com dois plugs em cada extremidade e se encaixam nos soquetes EAR e MIC do gravador (gravadores mais baratos têm mais chances de ter este tipo de soquetes). Se o seu gravador cassete estiver equipado com uma saída de áudio simples, ele tem grande chance de ser compatível com o computador.

Um contador no gravador pode ser muito útil; este dispositivo conta quantas rotações a fita dá. Use os números que são mostrados para localizar programas em fitas de longa duração. Entretanto, freqüentemente os contadores não são muito precisos, e assim use o contador somente para determinar a posição aproximada do programa.

A Fita Cassete

Você pode usar qualquer tipo de fita, embora deva evitar as muito baratas se der valor aos programas que estiver guardando. As fitas destinadas a computadores (C-10 ou C-12) funcionam melhor.

Use os menores cassetes possíveis. Comparada com a memória do seu T/S 1000, a fita guarda muita informação. Uma vez que toda a memória do seu computador leva menos de um minuto para ser escrita no cassete, os seus programas serão achados mais facilmente se houver apenas um programa em cada lado da fita.

Cassetes de duração muito curta (5 minutos de cada lado) podem ser conseguidos em lojas de aparelhos eletrônicos ou revendedoras de computadores.

Você terá muito menos problemas se usar fitas novas. Muitos gravadores não apagam a fita muito bem e o barulho deixado pelas gravações anteriores podem causar problemas.

Como segurança para o caso de uma fita ser estragada ou acidentalmente apagada, é uma boa idéia ter cópias de programas importantes. Uma das próximas seções deste livro trata de cópias de fitas. Uma vez o seu programa gravado, tenha muito cuidado com a fita. Mantenha-a afastada de campos magnéticos, de ímãs e de sua TV.

Programas Prontos

Os predecessores do Timex/Sinclair 1000, o Sinclair ZX-80 e ZX-81, foram os computadores mais vendidos no

mundo. Conseqüentemente, grande variedade de programas foi desenvolvida para o T/S 1000. Existem programas financeiros, educacionais, gráficos e, evidentemente, jogos destinados ao computador.

Existem grupos de usuários do Sinclair pelos Estados Unidos e por todo o mundo. Uma vez lançado no mercado, o computador Timex em pouco tempo formou grupos de usuários em quase todas as cidades. Estes grupos constituíram-se em clubes que existem para trocar informações sobre computadores. Além da troca de informações, sugestões de programações e outros tipos de conhecimentos, os membros são incentivados a trocar programas. Muitas organizações comerciais (incluindo Timex e Sinclair) vendem programas; em qualquer caso, as fitas gravadas anteriormente são largamente difundidas.

Algumas revistas existem somente para os usuários dos computadores Timex e Sinclair. Elas, freqüentemente, analisam os programas comerciais. Infelizmente, como existem muitos programas e poucas revistas, muitos dos bons programas não são analisados.

Se você comprar programas ou os obtiver de alguém, procure assegurar-se de que o programa foi escrito para o Timex/Sinclair 1000 ou para o ZX-81. Cada computador tem as suas próprias peculiaridades; logo, programas escritos para outros computadores não rodarão no T/S 1000 — o BASIC de cada empresa é único. Mesmo os progra-

mas escritos para o Sinclair ZX-80 (o modelo primitivo do ZX-81) podem não rodar no Timex/Sinclair.

A capacidade de memória de programas é uma consideração importante na compra de software. O seu Timex/Sinclair 1000 contém uma memória usável de pouco mais de 2000 bytes. Um *byte* é uma unidade básica de informação (com a letra A). Em termos computacionais, 2000 bytes de memória livre são denominados de 2K de RAM (random access memory) (memória de acesso randômico). O Sinclair ZX-81 vem com um 1K de RAM ou seja, somente um pouco mais da metade da memória do T/S 1000. Conseqüentemente, os programas escritos para um Sinclair ZX-81 comum funcionam no seu computador Timex. Entretanto, muitos programas são escritos especialmente para o ZX-81 ou para o Timex/Sinclair com memória extra. A extensão de memória mais comum aumenta a memória para 16K; isto significa que os programas maiores podem ser usados no computador. Os programas escritos para 16K de memória não funcionam em seu computador Timex sem a expansão de memória (veja o Apêndice A, para maiores informações). Se você comprar programas em fita, em uma loja ou pelo correio, o anúncio ou o "pacote" deve estipular a quantidade de memória necessária. Se um anúncio não estiver claro e se você não possuir uma expansão de memória, tenha cuidado ao comprar o programa. Se você estiver recebendo o software de um amigo ou de um grupo de usuários, economizará muito tempo se antes souber se o programa é compatível com seu computador.

Uma vez que você tenha a fita com o programa, colocá-la no computador é como reproduzir qualquer outro programa do gravador. Veja a seção "Reprodução de Programas da Fita", ainda neste capítulo, para você saber como proceder.

Você deve sempre fazer uma cópia de qualquer programa que comprar e depois guardar a fita original longe de equipamentos eletrônicos e campos magnéticos. Veja a seção "Como Copiar uma Fita", ainda neste capítulo.

Como Guardar Programas

Digite um programa em seu computador (para isto, use um pequeno programa); por exemplo, experimente o programa calculado no Capítulo 11). Uma vez reproduzido, **LISTE** o programa para ter certeza de que ele realmente está ali. Agora, volte a fita ao início. Coloque o gravador em RECORD. Então, usando o microfone, grave o nome do programa na fita. Gravando o nome do programa na fita você estará apto a achar o programa, mais tarde.

Você recebeu, com seu computador, um cabo curto com dois plugs na extremidade. Os quatro plugs têm duas cores diferentes. Encaixe qualquer plug no soquete MIC de seu gravador e a *outra extremidade de mesma cor* no soquete MIC do computador. Estes soquetes estão próximos do plug da alimentação. *Não* encaixe ainda a outra parte do cabo nos soquetes EAR.

Ajuste o volume do gravador em três quartos do máximo. Se o seu gravador tiver controle de volume automático, não se preocupe com o volume. Se o gravador tiver controles de graves e agudos, coloque o grave na posição mais baixa e o agudo na posição mais alta.

Agora você está pronto para guardar o seu programa na fita.

Digite o seguinte comando no computador (mas não tecle **ENTER**, ainda):

SAVE — "nome do programa"

A palavra-chave **SAVE** é gerada teclando S quando o cursor **K** está na tela. Use qualquer nome, mas escolha um que diga o propósito do programa. Por exemplo, se você estiver usando o programa *A Melhor Calculadora*, do Capítulo 11, use "Calculadora" como nome do programa. O nome pode conter espaços. Não use caracteres em modo inverso, no nome.

Agora coloque seu gravador em RECORD; se você está no início da fita, deixe-a correr por 5 segundos, antes que você digite **ENTER**. Esta demora permitirá que a fita em branco, no final, (chamada condutor) passe. Então, pressione **ENTER**. A tela da TV ficará em branco por alguns segundos, e depois ficará coberta por listras brancas e pretas. Após alguns segundos, a tela ficará branca novamente e o código de informação 0/0 aparecerá no canto inferior esquerdo, indicando que tudo correu bem. Desligue o gravador. Digite este comando (mas não pressione **ENTER** ainda):

SAVE "nome do programa"

Use o mesmo nome. Ligue novamente o gravador e tecle **ENTER**. Quando o 0/0 aparecer, repita o processo uma terceira vez. É uma boa idéia fazer duas ou três cópias. Ocasionalmente, devido a um acidente ou defeito na fita, uma ou duas delas podem ficar estragadas. O computador não tem capacidade de ler fitas com defeitos, mesmo que eles sejam apenas parciais; logo, havendo três cópias, você deixará de ter muitos problemas.

As fitas têm travas atrás de suas caixas, perto dos cantos. Se elas forem removidas, nada pode ser gravado na fita. Isto é uma forma de proteção permanente de seus programas, contra apagamentos acidentais de seu gravador. Se você mais tarde decidir regravar a fita, simplesmente coloque fita adesiva por cima dos buracos onde as travas estavam.

Leitura de Programas da Fita

Volte a fita, desconecte o cabo do soquete do MIC, e então coloque-o no soquete EAR (ou MONITOR) do gravador, e o outro plug de mesma cor no soquete do computador. Você não deve ligar o cabo nos soquetes EAR e MIC, ao mesmo tempo. Muitos gravadores não funcionam bem com o computador, se ambos os soquetes EAR e MIC estiverem conectados.

Digite o seguinte comando (mas não tecle **ENTER** ainda):

LOAD "nome do programa"

O nome que você der tem que ser exatamente o mesmo que usou quando gravou o programa. Se o nome original contiver espaços, este deverá tê-los também.

Agora coloque o volume do gravador em três quartos do total e tecle **ENTER.**

A tela mostrará linhas diagonais por poucos segundos e a seguir aparecerão, por alguns segundos, linhas horizontais que são provocadas pela versão gravada do programa.

Quando as linhas horizontais desaparecem, a tela da TV deve voltar a ficar limpa, exceto pelo 0/0 no canto inferior esquerdo. Se as linhas diagonais voltarem, o computador não foi capaz de ler a cópia da fita. Deixe a fita continuar a rodar, para ver se a segunda ou terceira cópias serão lidas.

Se nenhuma das três cópias funcionarem ou se as linhas horizontais nunca aparecerem, tecle **BREAK** (a tecla do **SPACE**) para impedir que o computador fique esperando, desnecessariamente, a leitura.

Se 0/0 apareceu, tudo correu bem. Agora tecle **LIST** para ver se o programa é, de fato, o que você quer. Se afirmativo, pule o resto desta seção.

Se, entretanto, o computador não conseguiu obter o programa da fita, pouco você pode fazer. Aqui estão os possíveis problemas e soluções:

- O volume pode não estar alto o bastante. Tente aumentá-lo. Se isto não funcionar, ele pode estar

muito alto e você deve tentar diminuí-lo (entretanto, é pouco provável que ele esteja muito alto).

- Cheque os controles de tonalidade (se estes existirem) para ver se o agudo está no máximo e o grave está no mínimo. Se houver só um controle, deixe-o no máximo.
- Os plugs podem não estar firmes ou podem estar nos soquetes errados. Para conseguir um programa da fita, o cabo deve estar no soquete EAR de ambos, no do gravador e no do computador. A mesma cor deve estar ligada em cada um. Esteja certo de que os plugs estão realmente ligados. Se o plug estiver solto ou muito apertado, o soquete EAR de seu gravador cassete pode precisar de um plug de tamanho diferente. Encontram-se adaptadores em lojas de eletrônica.
- O gravador pode ser estereofônico. Os monofônicos trabalham com maior eficiência na leitura e gravação das fitas.
- Você pode ter usado todos os quatro plugs durante a gravação. Isto pode causar interferências na fita. Tente regravá-la.
- Se você está usando uma fita que já tenha sido gravada, ela pode ter sido feita para computadores diferentes. A fita deve ser feita para o Timex/Sinclair, contendo uma extensão de memória. A menos que você tenha uma extensão de memória, somente programas destinados ao Timex/Sinclair 1000 ou ZX-81 podem ser usados sem ela. As especificações do programa devem

dizer qual a memória necessária. Se você não tiver uma extensão de memória, o seu computador pode aceitar apenas 1 ou 2K em programas.

- A fita pode ter sido gravada incorretamente. Alguns problemas podem ser detectados, se você escutar a fita. Desconecte o cabo do gravador e volte a ela. Abaixe o volume e escute. Você deve escutar a voz anunciando o nome do programa (se você o gravou) e depois um zumbido leve. Após isto, se seguirão cinco segundos de silêncio. O silêncio significa que o computador espera o programa. O gravador pode estar gerando muito barulho, ou a própria fita pode ter muito chiado. Você pode tentar reproduzir a fita com o volume um pouco mais baixo. Depois do silêncio vêm alguns segundos em um tom alto, desagradável e estridente. Isto é o programa, depois do qual o ruído suave deve voltar a ser escutado. Se você não conseguiu esta seqüência de sons, o programa provavelmente não foi gravado. Cuidadosamente, grave novamente o programa. Se ele não funcionar ainda, os soquetes do computador podem estar com defeito. Tente outro gravador. Se você não conseguir fazer nenhum gravador funcionar, há grande probabilidade de algo estar errado com o seu computador ou com o cabo. Examine o cabo, por precaução.

Se você colocou um programa na fita mas não ouviu a seqüência certa, de sons, quando o escutou, volte a fita e tente outra cópia.

- Se você estiver perto de um rádio amador ou uma antena comercial de rádio (o que pode estar criando interferência), tente mudar seu computador para outra parte da casa.
- Alguns gravadores cassetes captam zumbidos da rede elétrica. Tente, se possível, usar pilhas ou baterias em seu gravador.
- A fita pode ter algum barulho que o gravador não consegue apagar. Tente usar fitas novas.
- As travas podem ter sido removidas da fita tornando a gravação impossível. Se isto tiver acontecido, coloque fita adesiva cobrindo os furos que as travas deixaram.
- Você pode não ter deixado a fita correr os 5 segundos do condutor. Regrave o programa, tomando cuidado para deixar o condutor passar, antes que a gravação comece.

Como Copiar uma Fita

É muito fácil copiar um programa, de uma fita para outra. Primeiro carregue o programa para o computador (veja a seção anterior, para saber como se faz). Coloque então uma fita nova no gravador e grave nela o programa. Você não pode copiar uma fita inteira de uma só vez; só se pode copiar um programa de cada vez.

Como há possibilidade de que um trecho da superfície da fita não funcione bem, é boa idéia fazer várias cópias do programa na fita nova.

Programas Auto-Executáveis

A declaração **RUN** limpa todas as variáveis da memória, antes de rodar o programa. Esta pode ser uma ação desagradável. Por exemplo, se um programa for planejado para manter um arquivo, ou se ele contiver informações que não são realmente escritas como parte do programa (mas previamente guardadas em variáveis), você pode executá-lo digitando **GOTO** - 0 (ou outro número de linha). Isto executa o programa sem afetar as variáveis.

Alternativamente, você pode fazer com que o programa grave a si próprio — variáveis e tudo — em cassete. A capacidade de salvar os valores contidos nas variáveis é um recurso de grande importância, do T/S 1000.

Veja aqui como fazer com que o **SAVE** possa ser usado como linha de programa. Quando esta linha for executada, o programa grava-se na fita (é claro que você deve ter conectado o cabo, preparado o gravador e começado a gravar). Você poderia usar declarações como esta:

```
100 SAVE "PROGRAMA"
200 GOTO 0
```

O comando **SAVE** será executado e depois o **GOTO** também; e o programa continuará correndo; gravado na fita desta maneira, ele começará a ser executado imediatamente após a declaração **SAVE.** Quando você ler o programa da fita, a declaração **GOTO** será executada imediatamente e o programa começará de novo. A decla-

ração após o **SAVE** não precisa ser o **GOTO**-0; pode ser qualquer declaração executável.

Como Tornar suas Fitas mais Simpáticas

Se você quiser facilitar o seu trabalho, é possível criar programas que automatizam o processo de leitura de programas. Por exemplo, você podia escrever um pequeno programa como este:

```
 100 PRINT "ENTRE COM O NÚMERO DO
     PROGRAMA"
 200 PRINT "1 "JOGOS-DE-DADOS"
 300 PRINT "2 "CHEQUES"
 400 PRINT "3 RECEITAS"
 500 PRINT "4 ARQUIVOS"
 600 INPUT A
 700 GOTO A*1000
1000 LOAD "JOGO DE DADOS"
2000 LOAD "CHEQUES"
3000 LOAD "RECEITAS"
4000 LOAD "ARQUIVOS"
```

Este "menu" poderia ser o primeiro de uma fita que também contenha os programas JOGOS DE DADOS, CHEQUES, RECEITAS e ARQUIVOS. Você leria o menu (com o comando **LOAD** - "MENU"), o executaria, faria

com que o gravador continuasse a rodar e entraria com o número do programa desejado. O computador iria ler a fita até que encontrasse o programa que você especificou.

Usando a técnica dos "Programas Auto-executáveis" (veja a seção anterior) e um controlador de cassete que pode ser comprado em algumas lojas de computadores, você pode criar uma fita que ficaria automatizada completamente. Deixe a fita correndo e digite primeiramente **LOAD**- "MENU". Quando o menu for lido, ele começa a ser executado e espera que você escolha um programa. Cada programa na fita termina com a declaração "**LOAD**-MENU", que recomeça o ciclo. A menos que use um controlador de cassete, você teria de parar, iniciar e voltar com o gravador no momento apropriado.

Infelizmente, não existe um modo fácil de passar informações de um programa a outro.

6

DICAS ÚTEIS PARA PROGRAMAÇÃO

Este capítulo consiste de pequenos programas que não fazem nada realmente útil, mas cada um ilustra um ponto importante da programação BASIC. Muitos tópicos abordados superficialmente nos capítulos anteriores serão tratados em detalhes aqui; e muitos outros novos, serão introduzidos. Tudo neste capítulo não será de muita valia, caso você pretenda seguir as explicações dos quatro capítulos de programação que virão a seguir. Considere as idéias deste capítulo como um "embelezamento" necessário para aprimorar suas habilidades em programação.

Modo SLOW e FAST

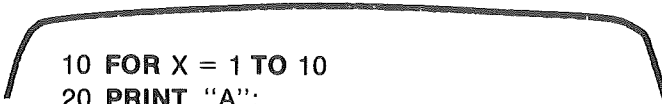
O T/S 1000 pode trabalhar em duas velocidades: **FAST** e **SLOW.**[10] Assim que você liga, ele está em modo **SLOW.** No modo **SLOW**, a tela da TV está constantemente atualizada: as informações novas que o programa escreve na tela aparecem imediatamente. No modo **FAST**, a tela é ignorada a não ser que o computador nada tenha para fazer.

Os programas que fazem muitos cálculos mas não imprimem muita coisa, funcionarão quatro vezes mais rápido em modo **FAST** que em modo **SLOW.** Por outro lado, apertando-se uma tecla em modo **FAST**, ela pisca. Além do mais, as informações da tela serão mostradas somente enquanto o programa estiver esperando uma entrada ou enquanto uma **PAUSE** estiver sendo efetuada.

O modo que você escolhe é, em geral, somente uma questão de gosto. Quando o programa necessitar de um modo ou de outro, nós o avisaremos.

Observe que se você está entrando ou examinando um programa que tem muitas declarações, pode querer trabalhar em modo **FAST.** No modo **FAST**, a imagem da tela é toda construída primeiro, e depois mostrada de uma vez só. No modo **SLOW**, a tela é mostrada à medida que

---

[10] Significam **RÁPIDO** e **LENTO**, respectivamente.

estiver sendo construída. A espera pode ser incômoda para algumas pessoas; logo, use o modo que preferir.

Tudo Sobre "PRINT"

Até aqui, a declaração **PRINT** lhe deve ser familiar. Esta seção discute os modos em que você pode usar itens na declaração **PRINT** para controlar o modo de como a tela é mostrada.

Você já viu como as vírgulas e os pontos-e-vírgulas mudam o modo de impressão. Para lembrar, quando um item do **PRINT** é seguido de um ponto-e-vírgula, o próximo item impresso aparece na mesma linha e imediatamente após o primeiro item.

Tente digitar e rodar este programa:

```
10 FOR X = 1 TO 10
20 PRINT "A";
30 NEXT X
```

Todos As aparecem lado a lado em uma mesma linha. Tente mudar o ponto-e-vírgula. Execute o programa de novo. Desta vez os As aparecem em duas colunas. Quando um item é seguido de uma vírgula, o próximo item a ser impresso aparecerá na outra metade da tela.

TAB e **AT** podem ser usados para mudar a posição onde o **PRINT** mostra seus itens.

Estes dois itens são muito parecidos. O **TAB** especifica uma coluna da linha corrente. O **AT** especifica a linha e a coluna da tela.

Em ambos os casos, a coluna é contada a partir da esquerda e é uma posição absoluta. Por exemplo, **TAB** 31 sempre se refere à coluna mais à direita da tela.

Quando seguidos por um ponto-e-vírgula, o **TAB** e o **AT** imprimem o próximo item do **PRINT** imediatamente após a posição especificada por eles.

A impressão, normalmente, ocorre de linha para linha quando a linha fica cheia. A posição do **PRINT** move-se para baixo da tela quando um item é impresso. O **TAB** não muda a direção da posição do **PRINT** para avançar transversalmente, ou para baixo da tela: ele só pode movimentar para frente a posição do **PRINT**. Se a posição do **PRINT** estiver na coluna 15 e você der um comando **PRINT TAB** 5, a posição do **PRINT** se moverá para a coluna 5 da próxima linha. O **PRINT AT**, entretanto, pode movimentar a posição da impressão para qualquer lugar da tela (o **PLOT** e o **UNPLOT** também podem mudar a posição de impressão para qualquer lugar da tela).

Os comandos **CLS, CLEAR, NEW, CONT** e **RUN** movem a posição de impressão para o canto superior esquerdo da tela. Também, quando qualquer programa ou comando direto acaba de ser executado, a posição de impressão volta ao canto superior esquerdo.

Tente digitar este programa:

```
100 FOR X = 20 TO 1 STEP -2
110 LET Z = 0
120 FOR Y = 30 TO 1 STEP -2
130 PRINT AT X, Y; "AT";
140 LET Z = Z + 1
150 PRINT TAB Z; "T"
160 NEXT Y
170 NEXT X
```

Rode este programa algumas vezes (ele deve ser rodado em modo **SLOW**). Observe que os Ts (impressos pela declaração com o **TAB**) começam a ser impressos nas linhas seguintes aos ATs. Quando o número do **TAB** é menor do que o número da coluna do **AT**, o T é impresso na coluna da linha que se segue à posição de impressão. Entretanto, quando os ATs e os Ts se cruzam, a posição de impressão tem um número de coluna menor que o dado no comando **TAB**; deste modo, os Ts são impressos na mesma linha que os ATs. Observe o programa com cuidado até que você possa ver isto.

Um comando adicional muda o display: o comando **SCROLL**. O comando **SCROLL** limpa a linha superior da tela e move toda a tela uma linha para cima.

Adicione esta linha ao programa (a palavra-chave **SCROLL** está embaixo da tecla B):

```
155-SCROLL
```

Agora rode o programa. Os Ts e os ATs correm em linhas diagonais subindo a tela, porque os valores do AT referem-se às posições absolutas da tela. Em outras palavras, após cada AT que for impresso, todas as linhas se movem para cima, mas o AT ainda fica impresso na mesma linha. Uma vez que o item anterior daquela linha seja movimentado para cima, forma-se um desenho em diagonal.

PLOT, UNPLOT e GRAPHICS

Quando você olhou os caracteres na última seção, deve ter notado que existiam algumas figuras cinzas e negras. São caracteres gráficos. Você os verá impressos em 20 teclas. Você também deve ter notado que cada um dos outros caracteres (exceto as palavras-chave) apareceram em duas formas: o caractere normal, negro, em fundo branco, e o inverso de um caractere branco, em fundo negro (o espaço inverso é um quadrado negro). Os cursores são exemplos de caracteres inversos quando o computador está em modo gráfico.

A tecla 9 tem o rótulo **GRAPHICS** em vermelho. Mantenha a tecla **SHIFT** pressionada e tecle 9. A palavra GRAPHICS não aparece; ao invés disto, o cursor L muda para o cursor G . Este indica que o computador está em *modo gráfico.* Agora, digite qualquer caractere; ele aparecerá em branco com fundo negro, tal como o cursor. Se

você digitar G, verá uma duplicata do cursor **G** . Tente digitar G, K e L. Estes caracteres são iguais aos três cursores. Se você digitar um S, ele se parecerá com um marcador de erro de sintaxe. Pressione a tecla **SPACE.** Um quadrado negro aparece. Estes são os caracteres inversos, sendo eles opostos aos caracteres que geralmente aparecem.

Agora mantenha o **SHIFT** pressionado e tecle todas elas. Quando o computador está em modo gráfico (isto é, o cursor **G** está ativo), pressionando-se **SHIFT** e digitando-se uma das teclas com caractere gráfico faz-se com que este apareça na tela. Se qualquer um dos caracteres for digitado sem a tecla **SHIFT**, o caractere inverso da tecla pressionada aparece. Se você pressionar **SHIFT** e pressionar qualquer tecla que não tenha caractere gráfico, o inverso do caractere em vermelho é impresso. Tente digitar a vírgula, com o **SHIFT** acionado, por exemplo.

Para deixar o modo gráfico, digite **SHIFT**/9; o cursor **L** retorna. Tecle **SHIFT**/P para fechar as aspas. Agora tente teclar **ENTER.** A string de caracteres gráficos (a que você digitou) será impressa no alto da tela, tal como você digitou dentro das aspas.

Você pode usar caracteres gráficos em declarações **PRINT** ou em strings, mas não em nome de variáveis.

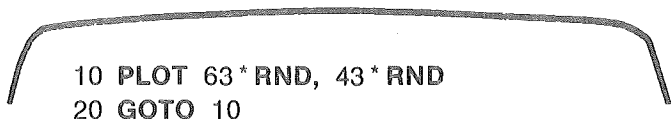
Um par de declarações importantes para a criação de imagens gráficas em sua TV: o **PLOT** e o **UNPLOT.** Digite este comando:

```
PLOT 10,10
```

Um pequeno quadrado, com um quarto do tamanho do caractere normal aparece na TV. O **PLOT** mostra simplesmente um destes quadrados na posição "10 por 10". O **UNPLOT** imprime um quadrado branco, apagando assim qualquer quadrado preto localizado na posição que você deu. Tecle **NEW** para limpar a memória e entrar com este programa (ele deve ser rodado em modo **SLOW**).

```
10 PLOT 20,20
20 UNPLOT 20,20
30 GOTO 10
```

Rode este programa (com o **RUN**). Ele produz um ponto brilhante na tela. Para parar o programa, tecle **BREAK**.

Para outro exemplo, entre com este programa:

```
10 PRINT "ENTRE COM UMA COLUNA"
20 INPUT X
30 CLS
40 PRINT "ENTRE COM UMA LINHA"
50 INPUT Y
55 CLS
60 PLOT X,Y
70 GOTO 10
```

Rode este programa e tente entrar com alguns números. Se você entrar com um número maior que 64 para a coluna e 43 para a linha, o programa pára porque estes números estarão fora da tela. Observe que os números vão do canto inferior esquerdo que é o 0,0 até o canto superior direito que é o 64, 43. É muito diferente das posições usadas no **PRINT-AT**, onde 0,0 é o canto superior esquerdo (além disso os números são duas vezes maiores, pois cada posição de um caractere pode conter quatro **PLOTs** — dois na horizontal e dois na vertical).

Observe também que tudo que for impresso imediatamente após o **PLOT** começa a ser impresso após a posição do **PLOT**; isto se deve ao fato de que a posição do **PRINT** é movida pelo **PLOT**.

O programa Desenhando, no Capítulo 8, usa o **PLOT** e o **UNPLOT** de um modo bem interessante.

Aqui está mais um programa usando **PLOT**. Ele deve estar em modo **SLOW**:

```
10 PLOT 63*RND, 43*RND
20 GOTO 10
```

Este irá continuar até que você o pare. Use a tecla **BREAK**, quando já estiver cansado.

Loops FOR/NEXT[11]

Freqüentemente, um grupo de declarações pode ser repetido um número específico de vezes. O BASIC executa isto com as declarações **FOR... NEXT.**

Vamos supor que você queira imprimir seu nome dez vezes. Poderia fazer isto com as seguintes declarações:

```
10 LET A = 1
20 IF A> 10 THEN GOTO 60
30 PRINT A, "MARIA"
40 LET A = A + 1
50 GOTO 20
60 STOP
```

Este programa, primeiramente, checa se A é maior que 10; se não for, imprime o valor de A e o seu nome pulando então para a linha 20. Este processo continuará até a linha 20, onde A é maior que 10. O programa, então, pára porque A é igual a 11.

Um loop **FOR/NEXT** funciona da mesma maneira que estas declarações (não se esqueça de que o **TO** é uma palavra-chave que você digita teclando **SHIFT/4**).

---

[11] **LOOP:** Sempre que nos referimos a um "loop", estaremos nos referindo a um laço **FOR/NEXT.** Lembre-se do looping de um avião. Veja pág. 27.

```
10 FOR A = 1 TO 10
20
40 NEXT A
50
```

O 20 e o 50 foram digitados sozinhos para eliminar as linhas de mesmo número do programa anterior; a linha 30 permaneceu como estava. Agora, dê **RUN.**

Os resultados são exatamente os mesmos da primeira versão deste programa.

Vamos voltar à primeira versão para ilustrar mais os loops **FOR/NEXT.** Digite estas declarações:

```
10 LET A = 10
20 IF A < 1 THEN GOTO 60
40 LET A = A-1
50 GOTO 20
```

Observe que desta vez o contador do loop começou em 10, e o seu valor diminuiu 1 de cada vez. Quando o valor de A fica menor que um, o programa termina.

Antes de olhar um loop **FOR/NEXT** equivalente, vamos experimentar mais um exemplo:

```
10 LET A = 0
20 IF A < 1 THEN GOTO 60
40 LET A = A + .2
```

Deixe o resto igual e rode esta versão do programa. Desta vez, o contador do loop começa em 0 e cresce de .2 até alcançar 1. O seu nome é impresso seis vezes.

O valor .2 do exemplo (-1 no exemplo anterior e 1 no anterior a este) é chamado de *incremento do loop.* Durante cada passo (ou cada incremento) dado pelo loop, o contador de loop é incrementado até que seu valor seja maior que o valor final definido na declaração **IF.**

A menos que você declare de outra maneira, o incremento numa declaração **FOR** é 1. Você também pode definir os valores inicial e final para serem os que quiser.

A forma comum de uma declaração **FOR** é:

FOR nome de uma variável = valor inicial **TO** valor final [**STEP** INCREMENTO]

A parte entre colchetes, a palavra-chave **STEP** e o incremento são opcionais. Se você não os der, o valor 1 será sempre usado. Se o valor do **STEP** for negativo, o valor inicial do contador do loop deve ser maior do que o valor final. Um incremento negativo requer que o valor inicial seja maior do que o valor final. Deste modo, você pode escrever programas que "contam para trás".

Os Comandos IF ... THEN e as Expressões Relacionais

Todas as declarações **IF ... THEN** têm a mesma forma. Se alguma expressão matemática for verdadeira **EN-**

TÃO execute este comando. Se a expressão matemática for falsa, execute a próxima instrução do programa.

Existem seis relações que podem ser testadas em uma declaração **IF ... THEN.** São elas:

EXPRESSÃO	**SÍMBOLO**
maior que	>
menor que	<
igual a	=
maior ou igual a	> =
menor ou igual a	< =
diferente de	< >

Você pode unir expressões, usando as palavras-chave **AND** e **OR** (**SHIFT**/2 e **SHIFT**/W).
Examine a seguinte declaração:

```
  65 IF (NÚMERO> TENTATIVA AND NÚMERO
10< = TENTATIVA)
  OR NÚMERO< TENTATIVA AND NÚMERO +
10> =TENTATIVA)
  THEN PRINT "MAS VOCE ESTA PERTO"
```

A declaração 65 é feita de etapas de expressões relacionais. Cada etapa das expressões é, por sua vez, formada por duas expressões separadas. Quando o computador executa a declaração 65, seu primeiro teste é ver se NUMERO é maior que TENTATIVA. Se for, o computador

checa se NUMERO diminuído de 10 é ainda maior (NUMERO-10< = TENTATIVA). Como as duas relações estão ligadas por um **AND**, as duas juntas serão verdadeiras somente se ambas forem verdadeiras.

Igualmente, o segundo conjunto é verdadeiro somente se ambas as partes forem verdadeiras. Contudo, o primeiro conjunto está ligado ao segundo pela palavra-chave **OR**; logo, se um dos conjuntos for verdadeiro, a declaração **IF** é verdadeira e o comando após o **THEN** é executado.

Em termos gerais, quando o **AND** une duas expressões, ambas devem ser verdadeiras para que as duas juntas possam ser verdadeiras. Quando o **OR** conecta duas expressões relacionais, se uma das duas for verdadeira (ou se ambas forem), então as duas juntas serão verdadeiras.

As condicionais podem também ser aplicadas a strings:

IF "JON" > "JOÃO" **THEN PRINT** "JON"

O computador imprime JON porque o código interno para JON é um número maior do que o código interno de JOÃO. O computador compara os nomes, letra por letra. As duas primeiras letras, J e O, são iguais em ambos os nomes. A terceira letra de JON é o N e a terceira letra de JOÃO é A. O código do N é 51, enquanto o código de A é 38. Logo, JON é maior do que JOÃO.

As strings podem ser comparadas, da mesma maneira que os números. Em todos os casos, a primeira letra (da

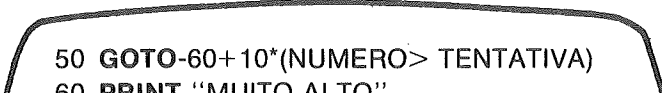
esquerda), onde as duas strings diferem, é o fator determinante.

Nós já falamos sobre como o computador reconhece o conceito de "verdadeiro" e "falso". Na verdade, o computador usa números para representar estes conceitos. Tente digitar este comando:

```
PRINT 1.5 > .9
```

O computador imprime um 1 no alto da tela.

Tente este comando:

```
PRINT 1.5 < .9
```

O computador imprime um 0 na tela.

Quando uma expressão for verdadeira, o computador assume o valor 1 no lugar da expressão relacional. Quando uma expressão for falsa, o computador assume o valor 0 para a expressão.

Você pode usar este artifício de várias maneiras. Por exemplo, tente digitar as seguintes declarações:

```
50 GOTO-60+10*(NUMERO> TENTATIVA)
60 PRINT "MUITO ALTO"
65 GOTO 20
70 PRINT "MUITO BAIXO"
75 GOTO 20
```

Rode este programa para ver que ele funciona da mesma maneira que o programa original (a linha que diz se você está perto foi omitida para fazer com que o exemplo fique mais simples).

O que faz o programa funcionar agora é a linha 50, a declaração **GOTO.** Se NUMERO for maior que TENTATIVA, a expressão NUMERO > TENTATIVA fica valendo 1. Como 60 + (10*1) = 70, o programa pula para a linha 70. Se NUMERO for menor que TENTATIVA a expressão NUMERO > TENTATIVA será igual a 0 e 60 + (10 * 0) = 60 de modo que o programa pula para a linha 60.

À medida que você for escrevendo mais programas, começará a descobrir que a pequena quantidade de memória do T/S 1000 é a grande limitação para que os seus programas possam funcionar. Técnicas como esta reduzem o tamanho de um programa e auxiliam você a fazer muito mais coisas em seu computador.

Funções

Algumas operações em um programa devem ser usadas freqüentemente. Ao invés de fazê-las a cada programa, para que este funcione, o T/S 1000 tem rotinas predefinidas chamadas *funções,* que executam estas operações.

Com uma exceção, o **RND**, todas as funções pegam um valor e retornam um outro que depende do primeiro. O primeiro valor é chamado *argumento* da função. Por exemplo, considere este comando:

```
PRINT INT A
```

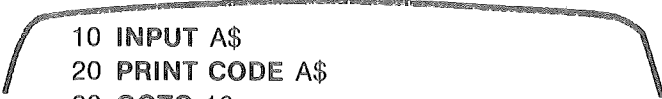
Aqui a variável A é o argumento da função **INT**.

O argumento de uma função pode ser uma variável, uma constante ou uma expressão. Entretanto, como as funções são calculadas antes dos operadores matemáticos (+, -, *,/ e **), você deve colocar a expressão entre parênteses. Por exemplo, tente digitar estes dois comandos:

```
PRINT INT (1 + .34)
PRINT INT 1 + .34
```

Na primeira, o computador soma 1 e .34 e, então, obtém o **INT** de 1.34. No segundo comando, o computador pega primeiro o **INT** de 1 e, então, adiciona o resultado a .34, resultando 1.34.

Muitas funções requerem argumentos strings. Estas funções são **CODEVAL** e **LEN**; todas produzem resultados numéricos. O **VAL** examina o conteúdo da string como se fosse uma expressão numérica. Por exemplo, digite as seguintes declarações (primeiramente tecle **NEW** para limpar a memória):

```
10 PRINT "COLOQUE UMA EXPRESSAO"
20 INPUT A$
30 PRINT VAL A$
40 GOTO 20
```

Rode o programa, entrando com qualquer expressão numérica válida. A expressão que você deu é guardada na variável alfanumérica (string) A$. O **VAL** de A$ é então impresso. Devido aos resultados, você pode imaginar que o **VAL** remove as aspas da expressão. Observe, entretanto, que o **VAL** não pode calcular uma string que não é válida como expressão numérica. As seguintes declarações, por exemplo, *não* iriam funcionar.

```
10 LET A$ = "ALO"
20 PRINT VAL A$
```

Para finalizar o programa, apague as aspas da esquerda e tecle o comando **STOP.**

A função **LEN** retorna o tamanho da string. Quando aplicado a uma string, o **LEN** retorna o número de caracteres da string.

O **CODE** retorna o número que é usado internamente pelo computador para representar o primeiro caractere de uma string. Todos os caracteres usados pelo computador são numerados de acordo com um sistema interno de códigos, indo de 0 a 255. O A, por exemplo, é 38. O B tem código 39. O caractere 1 tem o código 29. Estes códigos são listados em um encarte do manual que vem junto com o computador. Experimente este programa. (Tecle **NEW** primeiramente para apagar a memória):

```
10 INPUT A$
20 PRINT CODE A$
30 GOTO 10
```

Você pode descobrir o código de qualquer caractere, com este programa. Observe que se você der mais de um caractere, todos os caracteres após o primeiro são ignorados. Para finalizar o programa, apague as aspas da esquerda e dê um comando **STOP**.

O oposto do **CODE** é a função **CHR$**. Digite este programa:

```
10 FOR X = 0 TO 255
20 PRINT CHR$ X;
30 NEXT X
```

Este programa imprime todo o conjunto de caracteres usados pelo T/S 1000. Os espaços ocupados pelo ponto de interrogação são caracteres não imprimíveis (**ENTER**, por exemplo) ou não usados. Observe que pela medida de conservação de memória, todas as palavras-chave são guardadas internamente com caracteres simples.

Talvez a função mais estranha seja o **INKEY$**. Esta função retorna um caractere normal que é a tecla que está sendo pressionada no teclado quando o **INKEY$** estiver funcionando. Se nenhuma tecla for pressionada, o **INKEY$** retorna uma string vazia. Veja o programa desenhando no Capítulo 8 como um bom exemplo do uso do **INKEY$**.

π (PI) é uma constante, não uma função, mas é mencionada aqui. Entre com o seguinte comando:

```
PRINT PI
```

Observe que quando você digita a tecla que tem o π (**SHIFT/ENTER** para conseguir o cursor **F**, e então a tecla M), PI é mostrado na tela. O valor impresso é simplesmente o valor de π, uma constante muito importante em matemática. Por exemplo, o próximo programa imprime o comprimento de uma circunferência. Você deve entrar com o raio do círculo (o raio é a distância entre o centro e a borda do círculo, isto é, a metade do diâmetro):

```
10 PRINT "DE O RAIO"
20 INPUT RAIO
30 PRINT 2*PI*RAIO
```

Duas outras funções são muito úteis: **ABS** e **SGN. ABS** retorna o valor absoluto do argumento. Isto significa que se um argumento for positivo, é dado o mesmo número. Se o argumento for negativo, o sinal negativo sai e o resto do número é o resultado. **SGN** simplesmente diz se um número é negativo, positivo ou nulo. Se o resultado for 1, o número é positivo; se for -1 o número é negativo; e 0, se for nulo.

As outras funções serão muito úteis na ocasião propícia. Após você usar este livro, pode dar uma olhada na descrição das funções no manual que acompanha seu computador para saber o que todas elas fazem.

Notação Científica

Ocasionalmente, você pode ver um número como este:

2.432902E + 18

Este número pode parecer estranho. Está escrito em uma versão do computador, de *notação científica,* que é um modo abreviado de escrever números muito grandes. O E no meio do número significa "vezes 10 elevado à potência de"; assim, o número é igual a:

2.432902 * 10 **18

Se você se sentir mais à vontade com a forma comum de exponenciação, aqui está o mesmo número, de novo:

$2.432902 * 10^{18}$

Já que você pode converter para a notação simples, movendo o ponto decimal 18 casas para a direita, esta é uma abreviação conveniente para escrever números imensos.

O número acima é igual a 2.432.902.000.000.000.000, um número suficientemente grande para ser compreendido facilmente. Quando lidamos com números desse tamanho, ao vê-los em notação científica eles ficam mais fáceis de serem entendidos. Além disso, economizam memória.

Esse tipo de notação é também usado quando um número é muito, muito pequeno. Observe no número E, acima, que o expoente é precedido de um sinal mais (+). Se ele fosse precedido por um sinal de menos (−), então você moveria o ponto decimal para a esquerda do número indicado. Por exemplo, entre com este comando:

```
PRINT 2** - 20
```

A resposta que o computador imprime é:

9.5367432E - 7

O que equivale a 0.0000095367432 que é um número muito pequeno.

Na versão de notação científica do computador, 5E-10 é igual a 5 * 1/10E10.

7

REVISÃO DE JOGO DE DADOS

Como Melhorar o Programa de Jogo de Dados

```
 70 PRINT "QUANTOS DADOS?"
100 INPUT A
200 CLS
300 IF A = 0 THEN-STOP
500 PRINT INT (1 + 6 * RND)
600 LET A = A-1
700 GOTO 300
```

Nosso velho amigo, o programa dos dados do Capítulo 4, não é ruim para o que você pretende usar. Você pode-

ria usá-lo para jogar dados, por exemplo. Porém, os seus amigos provavelmente não ficarão impressionados. Afinal, a tela da TV mostra apenas um par de números. Seria uma demonstração muito melhor se o programa exibisse a imagem dos próprios dados.

Não é necessário começar a fazer tudo de novo. Seu programa de jogo de dados tem uma boa rotina central à qual podemos acrescentar muita coisa. Vamos reescrever nosso plano de três pontos, adicionando alguns pontos novos para realçar o programa.

1. Descobrir quantos dados o jogador quer "rolar". Se o jogador entrar o zero isto indica o fim do programa.
2. O programa escolhe um lado do dado, aleatoriamente.
3. Repete o passo 2 o número de vezes indicado pelo jogador, no passo 1.
4. Mostra graficamente os lados do dado, na tela.
5. Retorna ao passo 1.

Já realizamos os três primeiros objetivos de nosso plano no Capítulo 4. Ao adicionar os novos objetivos, vamos aplicar uma nova técnica de programação: o loop **FOR ... NEXT.** Algumas explicações serão dadas aqui, mas o aperfeiçoamento será mais facilmente entendido se você já tiver lido o capítulo anterior.

A maneira mais óbvia de melhorar o programa é recolocar a linha 600 com o eficiente loop **FOR ... NEXT.** Como você verá, isto também permite satisfazer o ponto 5 do

nosso plano. Uma declaração **FOR** diz: "Execute as seguintes declarações um certo número de vezes". O **NEXT** define o fim do loop.

Aqui estão as declarações no programa:

```
400 FOR X = 1 TO A
600 NEXT X
700 GOTO 70
```

A declaração **FOR** cria o loop. A variável X, chama-se *contador do loop.* Você pode selecionar qualquer letra simples, como contadora do loop. Quando o computador executa a declaração **FOR** pela primeira vez, ela cria a variável X (se uma já existir, é destruída), e então coloca seu valor igual ao número, após o sinal de igual (neste caso 1). Este é o valor inicial do contador do loop. O computador então executa o resto do programa até alcançar o **NEXT**-X. Neste ponto o computador adiciona 1 ao valor de X e verifica se o X alcançou um valor maior que A. Se X não for maior que A, o programa continua o loop.

Os contadores do loop são um tanto diferentes de outras variáveis. Eles sempre devem ter um nome com apenas uma letra. Você pode usar seus valores no meio do loop (na verdade, esta é uma das coisas mais poderosas dos loops **FOR ... NEXT**), mas você não deve mudar seus valores.

A simples adição destas três linhas nos permite executar o ponto 5. Agora, a execução do programa vai, automaticamente, para a linha 700, e depois desta o jogador

entra mais uma vez com um número e também dá-se a oportunidade de terminar o programa teclando 0.

Agora é hora de atacar o problema da apresentação gráfica. Primeiro, entretanto, devemos parar para aprender sobre uma parte muito importante da programação: matrizes.

Uma *matriz*[12] é simplesmente um grupo de variáveis, todas conhecidas por um mesmo nome. A declaração:

DIM - D$ (6,9)

cria uma matriz contendo seis variáveis strings: D$ (1), D$ (2), D$ (3), D$ (4), D$ (5) e D$ (6). Cada variável na matriz chama-se *elemento da matriz.* O segundo número da declaração **DIM,** 9, é o tamanho de cada elemento da matriz.

Um elemento de uma matriz string é exatamente igual a qualquer variável string dentro dela. Entretanto, quando você define uma variável string ou uma matriz com uma declaração **DIM,** a variável ou elemento da matriz permanece com o mesmo tamanho. Se você colocar uma string muito longa no elemento, este irá aceitar somente o que ele puder conter — o resto da string é perdido.

Você pode usar uma declaração **DIM** para dar um tamanho fixo à sua variável simples. Neste caso, você usa um comando como **DIM**-A$ (5). Após isto, o A$ sempre conterá somente cinco caracteres.

As matrizes numéricas são diferentes das matrizes strings, na medida em que você nunca define um "tamanho" para uma matriz numérica. Matrizes numéricas são

---

[12] Pode-se usar o nome *array* para designar uma matriz.

exatamente como variáveis numéricas comuns, exceto que elas têm, como nome, somente uma letra simples.

DIM-D$ (6,9) define uma matriz string com 6 elementos. Cada elemento pode conter 9 caracteres (por comparação **DIM** X (6) define uma matriz numérica chamada X com 6 elementos. Como sempre, cada elemento pode conter um número, embora este número possa ser muito grande).

Aqui estão as linhas para criar nosso gráfico:

```
 5 DIM D$ (6,9)
10 LET D$ (1) = "****0****"
20 LET D$ (2) = "***0*0***"
30 LET D$ (3) = "0***0***0"
40 LET D$ (4) = "0*0***0*0"
50 LET D$ (5) = "0*0*0*0*0"
60 LET D$ (6) = "0*00*00*0"
```

A linha 5 cria uma matriz alfanumérica com 6 elementos, cada um podendo conter 9 caracteres. As seis atribuições no programa formam strings para representar os dados. Observe atentamente as strings. Se três caracteres são pegos de uma vez e colocados um em cima do outro, cria-se uma imagem e parece-se com a face de um dado: isto é o que a string na linha 30 parece "cortadas e encaixadas":

```
0** ← os primeiros três caracteres de D$ (3)
*0* ← os caracteres do meio de D$ (3)
**0 ← os 3 últimos caracteres de D$ (3)
```

Mas você não digitou nada no programa para imprimir o dado. Se você acrescentar as seguintes declarações, o programa funcionará. (Tecle a palavra-chave **STEP** com **SHIFT**/E. Tecle a palavra-chave **TO** com **SHIFT**/4.)

```
500 LET B = INT (1 + 6 * RND)
510 FOR Y = 1 TO 9 STEP 3
520 PRINT D$ (B, Y TO Y + 2)
530 NEXT Y
540 PRINT
```

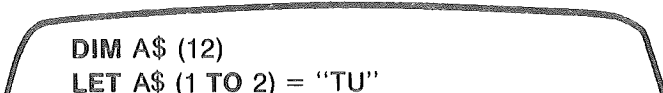
A declaração 500 é quase igual à velha, exceto que o valor aleatório é salvo em uma nova variável, B, ao invés de ser impresso diretamente.

A declaração 510 é parecida com o loop **FOR** que já tinha visto. A diferença é a palavra-chave **STEP.** Com o **STEP,** a declaração **NEXT** adiciona à variável do loop o número seguinte ao **STEP.**

A declaração 520, o **PRINT,** é a declaração-chave nesta nova versão do programa. Aqui está ela novamente:

```
520 PRINT D$ (B, Y TO Y + 2)
```

O B é o valor randômico entre 1 e 6, escolhido na linha 500. No loop inteiro, a declaração **PRINT** imprimirá o conteúdo do elemento da matriz nas linhas 510-530. Se você prestar atenção à declaração **FOR,** verá que ele dará três "voltas". A primeira variável do loop será igual a 1. A segunda, a variável do loop, será igual a 4; e a terceira volta, o Y, terá o valor 7.

Conseqüentemente, quando o **PRINT** for executado usará os seguintes valores:

```
PRINT D$ (B, 1 TO 3)
PRINT D$ (B, 4 TO 6)
PRINT D$ (B, 7 TO 9)
```

Se você dividir mentalmente as strings em três partes para ver como elas se unem para formar as faces do dado, a expressão "NUMERO-TO-NUMERO" na declaração **PRINT** pega as divisões da string. Esta técnica de divisão pode ser usada até mesmo do lado esquerdo do sinal de igual, de uma declaração **LET**.

Por exemplo, digite isto:

```
DIM A$ (12)
LET A$ (1 TO 2) = "TU"
LET A$ (3 TO 5) = "DO"
LET A$ (6 TO 9) = "BELE"
LET A$ (10 TO 12) = "ZA?"
PRINT A$
```

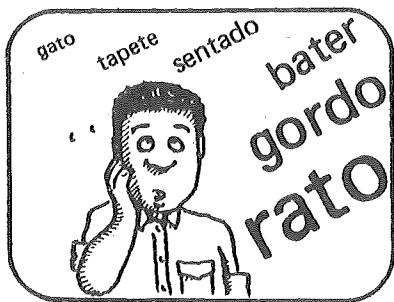
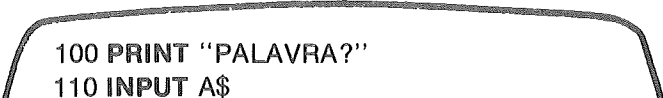
A mensagem TUDO BELEZA? é escrita na TV.

A declaração 540, onde tem um **PRINT** sozinho é usada para deixar um espaço em branco entre as faces de dois dados.

Vamos fazer um desafio de programação para você: tentar melhorar o programa de dados, fazendo com que eles apareçam em uma posição aleatória na tela.

8

PROGRAMAS DE JOGOS

Palavras Cruzadas:

Se você gosta de resolver palavras cruzadas, provavelmente há dias em que não consegue encaixar uma letra para completar a palavra. Este pode ser o programa de que você vai precisar naqueles dias.

Entre com uma palavra colocando um hífen onde está a letra desconhecida. O programa, então, randomicamente adiciona letras e imprime os resultados na tela. Isto se repetirá até que a tela fique cheia. Se você quiser continuar, tecle **CONT** e vai obter, então, outra tela repleta de palavras.

Execute o programa em modo **SLOW**. Ele funciona relativamente rápido e você estará vendo, com clareza, a palavra que deseja, se você estiver olhando enquanto o computador imprime as palavras. Se examinar cuidadosamente cada vez que a tela ficar cheia, você achará, com certeza, a palavra de que precisa.

Se a sua palavra tem apenas uma letra desconhecida, então você terá 26 combinações e o programa correrá bem; e se forem 2 letras, existem 676 combinações. Se houver 3 letras desconhecidas, existem 17.576 combinações e não há muita chance de ver qual a que você precisa. Vamos sugerir uma modificação para contornar este problema; primeiramente, vamos ver o programa Basic:

```
100 PRINT "PALAVRA?"
110 INPUT A$
120 LET B$ = A$
130 FOR X = 2 TO LEN A$
```

```
140 IF-B$ (X) = " — " — THEN LET-B$(X) = CHR$ INT
(CODE "A" + RND*26)
150 NEXT X
160 PRINT B$ + "  ";
170 GOTO 120
```

O programa guarda a palavra original em B$ e depois altera a cópia de B$ no loop **FOR/NEXT**. Quando um hífen (-) é achado em B$, ele será trocado por uma letra escolhida aleatoriamente. Este processo de escolha é o coração do programa e tem de ser analisado com mais atenção. A expressão:

CHR$ INT (CODE "A" + **RND** * 26)

é usada para escolher uma das 26 letras do alfabeto. A expressão entre parênteses seleciona um número entre 38 (que é código interno de A) e 63 (o código interno de Z). Como o **RND** produz sempre um número entre 0 e 1 (o valor pode ser às vezes igual a 0, mas sempre é menor que 1); multiplicando o **RND** por 26, teremos um valor entre 0 e 26 e pode ser algumas vezes 0, mas sempre será menor que 26. Adicionando o resultado ao valor do **CODE** "A", que é 38, produz-se um número entre 38 e 63.99999999 (usando o **CODE** "A", ao invés do número 38, salva um pouco de espaço e faz com que o objetivo da expressão fique claro). Obtendo o **INT** do resultado, teremos um número entre 38 e 63, inclusive. A função **CHR$** nos dá o caractere com o código contido no argumento de **CHR$**.

Conseqüentemente, a expressão nos dá uma letra escolhida ao acaso.

A linha 160 imprime as palavras obtidas, adicionando espaços em branco entre as expressões.

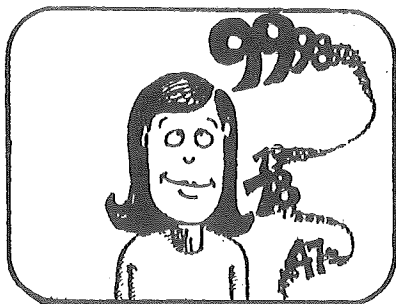
Como tínhamos dito anteriormente, este programa é prático se a palavra tiver uma só letra desconhecida ou se você não tiver a menor idéia de qual é a palavra. Ele pode então obter a palavra, ou pelo menos dar algumas idéias.

Se você reduzir o número de escolhas para as letras desconhecidas, aumenta as chances de o programa achar a palavra.

Você pode adicionar estas linhas ao programa:

```
115 PRINT "LETRAS?"
116 INPUT L$
140 IF A$ (X) = "—" THEN LET B$ = L$ (INT (1 + RND *
LEN L$)
```

A nova linha 140 é como a antiga, exceto pelo modo como a letra é escolhida. Aqui, um número randômico é escolhido entre 1 e o tamanho da string que você digitou (a função **LEN** retorna o tamanho de uma string). Logo, a expressão L$ (**INT** (1 + **RND** * **LEN** L$)) escolhe uma das letras que você digitou na linha 116. Escrever o programa desta maneira seria vantajoso especialmente se, por exemplo, você soubesse que a palavra só tem as vogais desconhecidas.

Adivinhe o Número:

Quando um computador parecer tão inteligente que você seria capaz de jurar que ele é inteligente, precavenha-se: a aparente "inteligência" é só o resultado de inteligentes declarações **IF ... THEN.**

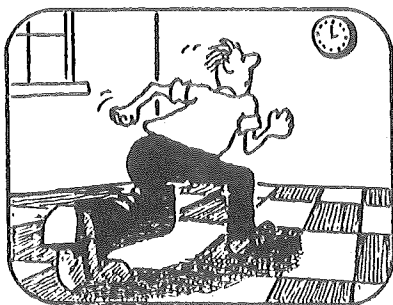
Estas declarações e outras que também usam expressões relacionais, permitem ao computador formar decisões. Elas são chamadas *expressões condicionais* e só dizem isto: "Se uma coisa for verdadeira, então faça isto."

Digite este programa. O **INT** e o **RND** são funções. Você deve pressionar **SHIFT/ENTER** para obter o cursor F e para poder digitá-las. Após digitar um **THEN** (**SHIFT**/3), o cursor K é impresso e você pode digitar os comandos **GOTO** ou **PRINT**. Cada declaração **IF** vai ultrapassar mais

de uma linha. Não se preocupe com isto; o computador não se preocupa com o tamanho de cada linha: ele simplesmente começará a imprimir uma segunda linha, quando você completar a primeira:

```
10 LET NUMERO = INT (RND * 100)
20 PRINT "SUA TENTATIVA?"
30 INPUT TENTATIVA
40 IF NUMERO = TENTATIVA THEN GOTO 80
50 IF NUMERO > TENTATIVA THEN PRINT "MUITO
PEQUENO"
60 IF NUMERO < TENTATIVA THEN PRINT "MUITO
GRANDE"
65 IF NUMERO > TENTATIVA AND NUMERO 10 < =
TENTATIVA) OR (NUMERO < TENTATIVA AND NUMERO
+ 10 > = TENTATIVA) THEN PRINT "MAS VOCE ESTA
PERTO"
70 GOTO 20
80 PRINT "VOCE VENCEU"
```

Tente executar isto. O programa escolhe um número inteiro aleatório entre 0 e 100. O jogador terá que adivinhar este número. Se o número não foi adivinhado antes que a tela ficasse cheia (o programa pára com o código de erro 5/20), dê um comando **CONT** (tecle C) para limpar a tela, e o programa vai continuar.

O Paradoxo de Zenão

O paradoxo lógico de Zenão é feito de modo que, ao andar pela sala, você deve, primeiramente, andar a metade do caminho, a seguir, mais a metade do caminho restante, e mais a metade do caminho restante, e assim por diante.

Teoricamente, se você continuar andando somente metade da distância, nunca chegará ao final!

Este programa calcula o tempo necessário para caminhar 100 pés com 1 passada de 1/4 de polegada. Para simplificar o problema, vamos considerar que se demora um segundo para atravessar cada metade da distância:

```
10 LET A = 100 * 12 * 4
20 LET B = 0
30 LET A = A/2
40 LET B = B + 1
50 IF A > 2 THEN GOTO 30
60 PRINT "DEMORA"; "B;" SEGUNDOS"
```

A linha 100 calcula quantos quartos de polegada existem em 100 pés. As linhas 30 e 50 formam um loop que divide a distância ao meio, adiciona um segundo ao tempo e depois verifica se esta é menor que um quarto de polegada (como a distância foi convertida para quartos de polegada, 1 é um quarto de polegada).

Não leva muito tempo. Ele acha que um quarto de polegada é suficientemente perto.

O menor número que o T/S 1000 pode trabalhar é um pouco menor que 1 E — 38. A letra E significa "vezes 10 elevado à potência de"; logo, este número é igual a 1 vezes 10^{-38} ou 00000000000000000000000000000000000001, um número bem pequeno, realmente.

Mude a linha 50 para:

50 IF A > 1E-38 THEN GOTO 30

Agora, execute o programa para ver quanto tempo ele leva para chegar ao término.

Monstro do Espaço!

Bem-vindo à frota espacial! Você recebeu uma árdua missão, perto do centro da galáxia, patrulhando um pequeno setor.

Este setor remoto é habitado por um monstro que vai engolir sua nave em um instante, se você deixar. Felizmente, você tem uma mina com a qual pode destruir o monstro. Infelizmente, o monstro é imortal. Quando você acha que destruiu o monstro, ele aparece novamente em algum lugar, em seu setor.

Como um problema a mais, a mina não coopera muito. Sua nave é muito pequena para carregá-la e então ela é teleportada para sua base, um asteróide em um canto do seu setor. O espaço tem um comportamento estranho nesta parte da galáxia: a mina não fica parada, e sim voltando à base. No início de sua patrulha, a mina volta à base quase que imediatamente. À medida que o tempo vai passando, a mina passa a cooperar mais e permanece por mais tempo em sua nave.

Você pode usar a mina para evitar o monstro, por algum tempo (se você for inteligente e com bastante sorte) mas sua única esperança é voltar à base onde a mina irá sempre proteger você. Entretanto, cuidado. O espaço perto de sua base é curvo: se você tocar na borda do setor, você pode não estar onde supõe e seus controles poderão agir estranhamente.

Aqui está o programa:

```
10 DIM X (2)
20 DIM V (2)
```

```
30 DIM S (2)
40 LET X (1) = 5
50 LET X (2) = 1
60 LET V (1) = 20
70 LET V (2) = V (1)
80 LET PLACAR = 0
90 LET TEMPO = 0
100 LET SEGURO = 0
200 CLS
210 PRINT AT X(1), X(2);"X"
220 PRINT AT V(1), V(2);"V"
230 IF V(1) = X(1) AND V(2) = X(2) THEN GOTO 1000
240 LET M$ = INKEY$
250 IF M$ = "5" OR M$ = "8" THEN LET V(2) = V(2) +
        SGN(VAL M$ - 7)
260 IF M$ = "6" OR M$ = "7" THEN LET V(1) = V(1) - 2
        * VAL M$ + 13
270 IF M$<>"S" THEN GOTO 300
280 LET S(1) = V(1)
290 LET S(2) = V(2)
300 IF INT(TEMPO/10)<>INT (RND * TEMPO/10)
        THEN GOTO 330
310 LET S(1) = 0
320 LET S(2) = 0
330 LET TEMPO = TEMPO + 1
340 PRINT AT S(1), S(2); "S"
350 IF V(1) = 0 AND V(2) = 0 THEN LET SEGURO =
        SEGURO + 1
360 IF SEGURO >= 10 THEN GOTO 1000
```

```
370 LET X(1) = X(1) + SGN(V(1)-X(1))
380 LET X(2) = X(2) + SGN (V(2)-X(2))
390 IF X(1)<>S(1) OR X(2)<>S(2) THEN GOTO 200
400 LET PLACAR + 1
410 LET X(1) = INT (RND * 22)
420 LET X(2) = INT (RND * 32)
430 GOTO 200
1000 PRINT "PLACAR"; PLACAR + SEGURO
```

Dê **RUN** no programa, em modo **SLOW**, usando as teclas com as setas para mover sua nave espacial (sua nave é o V). Não pressione **SHIFT** para usar as teclas das setas, como você faz normalmente; tecle as setas sozinhas. Sua base está no canto superior esquerdo e você pode ver a mina ali: é aquele símbolo gráfico. Para obter a mina para a proteção de sua nave, tecle S. Você não pode se mover e recuperar a mina ao mesmo tempo; logo, deve deixar de apertar as setas enquanto pressionar o S. A mina virá para sua posição e ali permanecerá (mesmo que movimente a nave) por um período de tempo que é muito pequeno no início do jogo, mas que fica maior à medida que você for jogando.

O monstro (o X) caça você. A velocidade dele é maior do que a sua quando ele se move em diagonal, mas é a mesma quando se move vertical ou horizontalmente. Você não tem muita chance de se manter afastado da criatura; tem que usar a mina para se proteger. Quando o monstro vai de encontro à mina, ele desaparece e reaparece em outro ponto da tela. Você faz um ponto cada vez

que o monstro bate na mina. Se fizer isto na base, o jogo lhe dá dez pontos e então termina. Você consegue dez pontos se terminar o jogo sem ser comido.

Aqui está o funcionamento do programa. Três matrizes X, V e S guardam as posições do monstro, de sua nave e da mina, respectivamente. As linhas antes da 200 iniciam as variáveis usadas no programa. TEMPO registra a duração do jogo. SEGURO registra o tempo em que você está seguro em sua base.

A tela é redesenhada a cada movimento para atualizar as posições. As linhas 210 e 220 imprimem você e o monstro. Se vocês estiverem na mesma posição, o monstro comeu você, e a linha 230 pula para o fim do programa.

A linha 240 obtém seu movimento, usando a função **INKEY$**. Esta função reconhece qualquer tecla pressionada no teclado.

A linha 250 vê se as teclas 5 ou 8 estão pressionadas. Estas são as teclas com as setas da direita e da esquerda. Se uma das duas for pressionada enquanto a linha 250 estiver sendo executada, a expressão SGN (**VAL** M$ — 7) é usada para determinar a nova posição. SGN retorna o valor − 1, se o argumento da função for negativo, e 1 se o argumento da função for positivo. O **VAL** converte uma string em um número com os mesmos caracteres e assim converte a string "5" no número 5, e a string "8" no número 8. Subtraindo 7 destes números, o resultado será um número negativo se a tecla pressionada for 5 (seta esquerda) ou um número positivo se a tecla for 8 (seta direita). Se a seta pressionada for a da esquerda, subtraí-

mos 1 da coluna; e se a seta for a da direita, adicionamos.

A linha 260 atua de modo semelhante, com as teclas de setas para cima e para baixo, embora use um modo diferente. Como a seta para cima é o 7 e a seta para baixo é o 6, o valor da tecla é dobrado e mudado de sinal, e soma-se 13 ao resultado. O 6 (seta para baixo) retorna um + 1 que é adicionado à coluna. O 7 (seta para cima) retorna um − 1 que também é adicionado à coluna.

A linha 270 verifica se a tecla pressionada foi o S. Se foi, a mina é movida para a posição atual da sua nave, colocando-se S(1) igual a V(1) e S(2) igual a V(2).

A linha 300 compara um número aleatório com o **INT** (TEMPO/10); se forem iguais, a mina volta à base. Após isto, 1 é adicionado a TEMP. A mina é impressa na linha 340.

A linha 350 verifica se sua nave está na base. Se estiver, então soma-se 1 à variável SEGURO. Se SEGURO for igual a 10, a linha 360 termina o programa.

As linhas 370 e 380 fazem com que o monstro persiga sua nave. A posição da nave é subtraída da posição do monstro, e a função **SGN** é aplicada ao resultado. O monstro, então, movimenta-se uma coluna e uma linha, em sua direção.

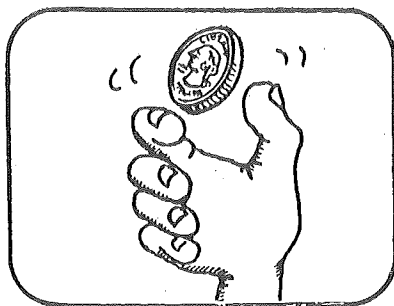
A linha 390 verifica se o monstro foi de encontro à mina. Se foi, você ganha um ponto e a posição do monstro é restabelecida randomicamente nas linhas 410 e 420.

A linha 430 volta para efetuar outro movimento.

Finalmente, a linha 1000 é a finalização do programa.

Ela imprime seu placar com o total de vezes que o monstro foi destruído pela mina e a quantidade de tempo que você ficou na base.

Uma observação: os movimentos ficam estranhos quando você bate no canto esquerdo e no alto da tela porque a "posição" da nave torna-se negativa. O **PRINT AT** usa o valor absoluto da posição dada; logo, se você der uma coordenada negativa, ela ficará positiva e as direções se inverterão. Se você for muito para baixo ou para a direita, o programa vai parar.

Cara ou Coroa?

A função **RND** (número randômico) no T/S 1000, como em todos os computadores, produz, na verdade, um número pseudo-randômico. "Pseudo-randômicos", ao invés de randômico, indica que é uma mera simulação de

um número randômico. O computador usa uma fórmula baseada geralmente no tempo que o computador fica ligado. Os números produzidos são aleatórios o bastante para a maioria de seus propósitos, mas podem não ser tão randômicos como alguns eventos físicos, como o de jogar uma moeda.

Quando você joga uma moeda, espera que saiam caras e coroas com a mesma freqüência. Isto não significa que *sempre* sairá a mesma quantidade de caras e coroas; tanto que, se jogar a moeda bastante vezes, você poderá esperar um número aproximadamente igual de caras e coroas. Digite este programa:

```
10 FAST
20 DIM B(2)
30 FOR X = 1 TO 10
40 LET A = 1 + RND
50 LET B(A) = + 1
60 NEXT X
70 PRINT B(1), B(2)
80 PRINT ABS (B(1) − B(2)) * 100/(B(2)+B(1));
          "POR CENTO"
```

O **RND** produz um número entre 0 e 1. Conseqüentemente, o número deve estar entre 0 e 0.5 na metade do tempo, e entre .5 e 1 na metade do mesmo período. Adicionando 1 ao **RND** na linha 40, temos como resultado um número entre 1 e 1.5 na metade do tempo e entre 1.5 e

2 na outra metade. Observe que a função **INT** não está aplicada ao **RND**, neste programa. Quando o computador espera um inteiro e obtém um número real, ele arredonda o número para o inteiro *mais próximo.* Isto é diferente de **INT** que remove a parte decimal e sempre nos dá um número menor que o original. Quando o computador arredonda um número, se a parte decimal for maior que .5 o computador repõe o menor inteiro maior que o número original. Se a parte decimal é menor que .5 então o computador retorna o maior inteiro menor que o número original, como **INT** faria.

A função **ABS**, usada na linha 80, retorna o valor absoluto de B(1)−B(2). Se o resultado for positivo, o **ABS** não faz nada, mas se o resultado for negativo, o **ABS** o transforma em positivo. A linha 80 dá a percentagem de diferença entre os dois números. Execute este programa algumas vezes.

Você terá números como 5 e 5, 6 e 4, 7 e 3. A diferença entre estes números é 0%, 20%, e 40%. Observe que a percentagem de diferença pode ser muito grande. Se você jogasse uma moeda só 10 vezes, não iria esperar que desse 5 caras e 5 coroas. Vamos tentar a mesma coisa com cem tentativas. Mude a linha 30. Você pode redigitá-la ou usar o comando **EDIT** para trocar 10 por 100.

A linha deve ficar assim:

30 **FOR** X = 1 **TO** 100

Tecle **ENTER** e nova linha 30 trocará de lugar com a antiga.

Rode o programa. Levará alguns segundos. Obtivemos 56 e 44 (12%) na primeira vez, 50 e 50 (0%) na segunda, 57 e 43 (14%) na terceira, 55 e 45 (10%) na quarta e 54 e 46 (8%) na quinta vez. Estes números estão muito próximos, mas pode ser interessante tentar este programa com 100 tentativas. Tecle **EDIT** para colocar a linha 30 na linha do **INPUT** (o cursor do **EDIT** permanece na última linha que foi adicionada, neste caso 30, a menos que tenha feito algo diferente).

Agora rode o programa novamente (leva 25 segundos para fazer 1000 tentativas; logo, seja paciente). Obtivemos 487 e 513 (2,2%) na primeira tentativa e 477 e 523 (4,6%) na segunda. Na terceira, obtivemos 511 e 489 (2,2%) e 517 e 483 (3,4%) na quarta.

Veja que, embora os números nunca sejam iguais, eles vão ficando mais próximos, à medida que aumentamos o número de tentativas. Isto é um bom sinal. Os números randômicos da vida real (moedas, dados, roletas) também têm esta flutuação. Daí o porquê de você poder acertar ocasionalmente um número na roleta de cassino. Entretanto, as probabilidades são poucas, de modo que o cassino leva vantagem sobre você. Se quiser, mude a linha 30 para:

```
30 LET A = 1.1 + RND
```

As chances serão no sentido de fazer com que o segundo número saia mais freqüentemente. É assim que os cassinos permanecem funcionando. Podem perder oca-

sionalmente, mas em 365 dias de 10 giradas de roleta e jogadas de dados, tudo fica a favor deles.

Se você quiser demonstrar que estes números ficam mais próximos um do outro à medida que as tentativas aumentam, tente mudar o número de tentativas para 10000 tentativas; obtivemos 5028 e 4972 que diferem somente 0,56%. À medida que o número de tentativas crescer, certamente os números ficarão mais próximos.

Desenhos

Este programa é uma imitação de um brinquedo onde você pode desenhar em um painel branco, manipulando dois botões.

Nesta versão, você usa as quatro setas nas teclas 5, 6, 7 e 8 para indicar a direção. Só digite as teclas, não use o **SHIFT.**

Como um melhoramento do brinquedo, você pode apagar as linhas, do mesmo modo que as desenhou. O programa começa em modo de desenho. Para apagar linhas, digite a tecla do **UNPLOT.** Você apaga depois com as teclas de direção, do mesmo modo que desenhou (porém, você só pode ver sua posição quando estiver atravessando uma linha PLOTADA).

Aqui está o programa:

```
 10 SLOW
 20 LT X = 25
 30 LET Y = X
 40 LET DESENHA = 1000
 50 LET APAGA = 2000
 60 LET C = DESENHA
 70 PLOT X,Y
100 LET A$ = INKEY$
110 IF A$ = "Q" THEN LET C = DESENHA
120 IF A$ = "W" THEN LET C = APAGA
130 IF A$ > = "5" AND A$ < = "8" THEN GOTO 100 *
          (VAL A$)
140 GOTO 100
500 LET X = X-1
510 GOTO C
600 LET Y = Y-1
610 GOTO C
```

```
 700 LET Y = Y + 1
 710 GOTO C
 800 LET X = X + 1
 810 GOTO C
1000 PLOT X,Y
1010 GOT 100
2000 UNPLOT X,Y
2010 GOTO 100
```

As declarações precedentes à linha 100 iniciam o programa, isto é, definem os valores iniciais das variáveis do programa. Um pixel (um quadrado com um quarto do tamanho do caractere) é impresso na posição 25. As variáveis DESENHA e APAGA são usadas para elucidar o programa. Estes valores são usados depois como acionadores dos modos DESENHA e APAGA. A variável C que está com o valor DRAW, diz se o programa está em modo DESENHA (**PLOT**) ou modo APAGA (**UNPLOT**). O programa é colocado em **SLOW** na linha 10 porque, simplesmente, não funcionará em modo **FAST**.

O loop formado pelas declarações 100 a 140 é o coração do programa. A função **INKEY$** retorna um caractere, se qualquer tecla estiver pressionada. Se nenhuma tecla tiver sido pressionada, **INKEY$** retorna um caractere nulo. As linhas 110, 120 e 130 só fazem alguma coisa se um Q, W, 5, 6, 7 ou 8 estiverem pressionadas. Q é a tecla do **PLOT**; W é a do **UNPLOT**. Quando Q é teclado, a variável C fica com o valor da variável DESENHA (que é tam-

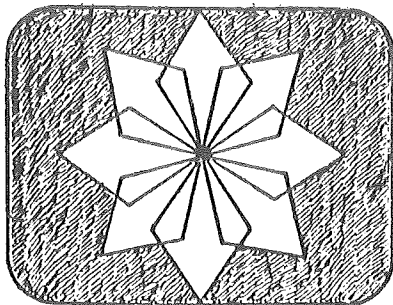
bém o valor inicial de C). Quando W é pressionado, C assume o valor da variável APAGA. A linha 130 verifica se o valor de **INKEY$** (A$) é igual a 5, 6, 7 ou 8, e então pula para uma linha baseada no valor de A$. A declaração que faz com que ele pule é determinada pela expressão:

100 * **VAL** A$

VAL retorna um valor numérico equivalente a string A$ (veja o Capítulo 6 para uma discussão detalhada). Se A$ contém um "5", **VAL** A$ retorna o valor numérico 5. Conseqüentemente, o comando **GOTO** 100 * **VAL** A$ pula para a linha 500 se a tecla 5 (seta esquerda) for pressionada, e assim por diante.

Cada linha 500, 600, 700 e 800 adiciona ou subtrai um dos valores de X ou Y, como for necessário. As linhas 510, 610, 710 e 810 pulam para a linha representada pelo valor de C. C pode ser igual a 1000 ou 2000 (DESENHA ou APAGA); logo, dependendo do modo, estas linhas pulam para as linhas 1000 ou 2000. A linha 1000 plota o ponto e a linha 2000 **UNPLOT** (APAGA) o ponto. As linhas 1010 e 2010 então, voltam para a linha 200 onde o loop continua até que outra tecla seja pressionada.

Observe que o programa poderia facilmente ser escrito com a linha 100 **LET** A$ = **INKEY$** trocada por um comando **INPUT.** Entretanto, a pessoa que estivesse usando o programa teria que teclar **ENTER** cada vez que desenhasse ou apagasse um ponto.

Figuras

Este programa toma um número dado por você e com ele cria figuras. Você deve entrar com um número de dois dígitos para rodar o programa corretamente.

Aqui está o programa:

```
10 PRINT "DE UM NUMERO"
20 INPUT N
30 PRINT N
40 FOR K = 1 TO N
50 LET R = 22 * K/N
60 LET A = PI * 100 * K/N
70 PLOT 32 + R * COS A, 22 + R * SIN A
80 NEXT K
90 GOTO 20
```

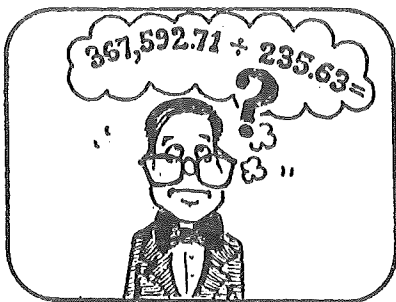
A "função" PI na linha 50 produz o valor 3.1415927. Você digita **PI** teclando **SHIFT/ENTER** para conseguir o cursor **F**, e depois, digitando a tecla M, que tem o símbolo do PI, π embaixo dela.

O programa pede seu número e a seguir usa-o para construir uma figura com base na combinação de funções trigonométricas. Uma vez a figura desenhada, o programa serve para outros dois números. A fim de parar o programa, entre com uma letra ou qualquer outro valor ilegal, enquanto o programa estiver construindo a figura.

Este programa funciona melhor em modo **FAST**.

9

PROGRAMAS EDUCACIONAIS

Exercícios de Aritmética

Este programa escolhe dois números entre 0 e 10, mostra um dos operadores aritméticos (+, −, * ou /), pergunta ao

usuário qual a resposta e então verifica se a resposta está certa. Se a resposta estiver errada, o usuário tem outra chance. Se estiver certa, será parabenizado.

O programa aqui apresentado é básico. Ele pode ser alterado, por exemplo, para números maiores que 10, usar exponenciação ou operações mais complicadas. Também pode informar se a resposta está correta, após algumas tentativas.

O segredo deste programa é a função **VAL**. **VAL** processa uma string como uma expressão numérica. O programa cria a string gerando dois números e usa o operador "+" para construir uma equação, usando dois números e um operador escolhido randomicamente. Tudo isto é feito na linha 200. Observe o programa:

```
 50 DIM D$ (4,1)
 55 LET D$ (1) = "+"
 60 LET D$ (2) = "-"
 65 LET D$ = "*"
 70 LET D$ (4) = "/"
 80 LET CERTO = 1000
 90 FAST
100 LET A$ STR$ INT (RND * 10)
150 LET B$ = STR$ INT (1 + RND * 10)
200 LET E$ = A$ + D$ (INT (1 + RND * 4)) + B$
250 IF INT VAL E$ < > VAL E$ THEN-GOTO-100
300 CLS
350 SLOW
400 PRINT E$; "=?"
```

```
 450 INPUT-A
 500 IF A = VAL E$ THEN GOTO CERTO
 650 PRINT "ERRADO. TENTE DE NOVO"
 700 PAUSE 60
 750 GOTO 400
1000 FOR X = 1 TO 5
1050 SCROLL
1100 PRINT AT 21,0; "********* PARABENS *********"
1150 NEXT X
1200 FOR X = 1 TO 20
1250 SCROLL
1300 NEXT X
1350 GOTO 90
```

A linha 80 parece estranha. A variável certa é usada para guardar o número da linha em que a mensagem de parabenização começa. A única razão para guardar um número de linha numa variável é tornar o comando **IF** na linha 500 mais claro. Você pode usar uma variável, uma expressão ou uma constante nos comandos **GOTO** e **GOSUB.**

As duas linhas que escolhem os números são diferentes. A primeira escolhe um número entre 0 e 9 e a segunda um número entre 1 e 10. Isto se faz para que o segundo número nunca seja 0. Caso contrário, poderia haver uma divisão por 0, o que provocaria erro.

A linha 250 verifica se a resposta é um número inteiro. Isto é arbitrário, uma vez que se pode ter respostas fracionárias. Por outro lado, um problema cuja resposta é

1/3 pode ter a resposta correta recusada pelo computador, por ser uma dízima.

Uma desvantagem deste programa é que não usa os sinais comuns para divisão e multiplicação. Outra desvantagem é que o usuário pode digitar a equação original e o computador dará como certa a resposta. É possível restringir a resposta a um número de dois dígitos, com essas modificações.

```
450 INPUT G$
460 IF LEN G$ > 2 THEN LET G$ = "999"
500 IF VAL G$ = VAL E$ THEN GOTO CERTO
```

Esta limitação funciona, pois o maior resultado que se pode ter tem dois dígitos. Caso a resposta tenha mais de dois dígitos, o computador dará a resposta como errada.

Outro ponto neste programa é o uso dos modos **SLOW** e **FAST**. Os cálculos são feitos no modo **FAST** (aproximadamente 4 vezes mais rápido). As respostas são mostradas no modo **SLOW**, para melhor visualização.

Se você quiser modificar os limites das equações geradas pelo programa, tudo o que precisa fazer é mudar algumas linhas. Por exemplo, para que o programa formule questões de apenas somas e subtrações, adicione as seguintes linhas:

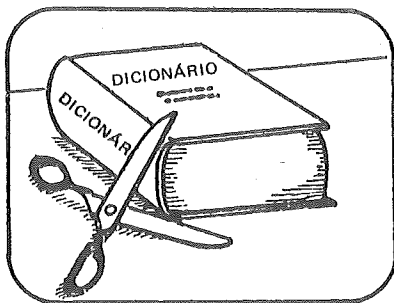
```
 50 DIM D$ (2,1)
 65
 70
200 LET B$ = STR$ INT (1 + RND * 2))+B$
```

(Digitando as linhas 65 e 70, sem nada, elas são retiradas do programa).

Se você quiser incluir a exponenciação no programa, basta aumentar o número de elementos em D$ para 5, mudando o 4 na linha 50 para 5, incluir uma linha para colocar (**) em D$ (5) e mudar 1 + **RND** * 4 na linha 200 para 1 + **RND** *5.

Se você quiser mudar os limites escolhidos, mude os 10s nas linhas 100 e 150 para qualquer número. Se você quiser colocar outras restrições, pode incluir comandos **IF**. Por exemplo, se quiser que os resultados sejam sempre positivos, inclua:

```
275 IF VAL E$<0 THEN GOTO 100
```

Vocabulário

Há uma técnica de ensino de línguas que consiste em escolher um texto, retirar algumas palavras e pedir para os alunos completarem as lacunas.

Este programa se encarrega de remover as palavras.

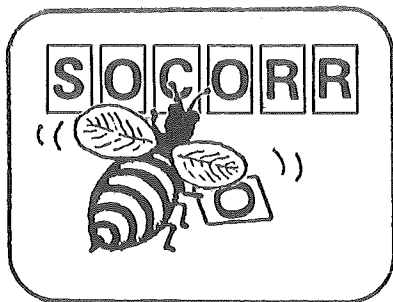
```
 10 LET C = 1
 20 PRINT "REMOVER QUANTAS PALAVRAS?"
 30 INPUT A
 40 PRINT "DIGITE O PARAGRAFO"
 50 INPUT A$
 60 CLS
 70 FOR X = TO LEN A$
 80 IF A$ (X) <>" " THEN GOTO 100
 90 LET C = C +1
100 IF C>A THEN LET C = 1
110 IF C<A OR A$ (X) = " " THEN GOTO 130
120 LET A$ (X) = "?"
130 NEXT X
140 PRINT A$
```

Como a variável string A$ não está dimensionada, seu parágrafo pode ter qualquer tamanho. Você não deve digitar **ENTER** ao final das linhas do texto, ou o computador entenderá que já deve começar o trabalho. Se isto "cortar" o seu trabalho, você pode, inclusive, incluir mais espaços; mas isto enganará o computador, fazendo com que a contagem das palavras seja incorreta.

A linha 80 procura espaços. Cada vez que ela acha um, o programa sabe que passou por uma palavra e o contador (variável C) é aumentado em uma unidade. Quando o contador se iguala ao valor das palavras a serem retira-

das, o programa procura a próxima letra e troca-a por um ponto de interrogação. O programa faz isto até encontrar outro espaço em branco indicando o final da palavra. Quando a linha 80 acha o espaço no final da palavra, C é incrementado de 1. Quando a linha 100 descobre que C (contador de palavras) é maior que A (número de palavras a serem retiradas), o programa rejeita o valor de C para 1, de novo, pois supõe-se que a contagem das palavras irá recomeçar. A linha 110 verifica se C é menor que A e pula a linha 120.

Você pode notar que a linha 120 seria pulada de qualquer maneira se a instrução **OR** A$ (X) = " " na linha 110 levasse à execução do comando **GOTO**. A cláusula **OR** é necessária para conter os espaços duplos. Se ela não estivesse aí, o segundo espaço seria substituído por um ponto de interrogação.

Soletrando

Este programa lê até 10 palavras digitadas e então você diz quantas letras devem ser retiradas (a palavra deve ter, no máximo, 10 letras). O computador, então, substitui as letras ao acaso e o usuário deve descobrir quais as letras que foram retiradas. Por exemplo: você digita "estável" e pede que duas letras sejam substituídas. Uma possível saída pode ser: EST? VE? O usuário deve descobrir qual é a palavra. Embora o programa dê ao usuário tantas chances quantas forem necessárias, a adivinhação prossegue indefinidamente. Se o usuário desistir, o programa imprime a resposta certa. Se ele acertar, há uma mensagem de parabéns.

Como este programa é razoavelmente grande, ele está listado em seções, com diferentes funções, cada. Eis aí a primeira parte:

```
 1 LET T = 10
 5 SLOW
10 DIM C$ (10)
15 DIM A$ (T, 10)
20 LET RT = 0
25 LET WR = RT
30 PRINT "ENTRE COM PALAVRAS"
40 FOR X = 1 TO T
50 INPUT A$ (X)
60 IF A$(X) < "A" THEN GOTO 90
70 PRINT A$ (X)
80 NEXT X
```

```
 90 CLS
100 PRINT "TIRAR QUANTAS LETRAS?"
110 INPUT R
```

O display parece melhor no modo **SLOW**; na verdade, a mensagem de parabéns não trabalha no modo **FAST**. A linha 5 dá o valor 10 à variável T. Esta variável é usada para definir o número máximo de palavras que podem ser usadas pelo programa. Note que ela também é usada para definir a matriz A$ (que guarda as palavras) e também na linha 40 onde ela define o valor para o término do loop. Também será usada adiante. Em cada caso ela está relacionada com o número máximo de palavras que o programa pode usar. Isto faz com que o trabalho de modificar o programa fique mais fácil. Se você quiser aumentar o número máximo de palavras, basta alterar o valor de variável T na linha 5.

A linha 15 **DIM** A$ (T, 10) limita o tamanho das palavras em 10. O segundo valor não é dado a variáveis e, por isso, é mais difícil modificar o tamanho das palavras do que seu número máximo. Como sempre, há uma preocupação em manter o programa pequeno, simples e fácil de ser alterado e consertado.

A linha 10 dimensiona a variável C$ em um tamanho fixo de 10. C$ é usada para receber as tentativas do usuário para soletrar corretamente as palavras. O valor de C$ é, então, comparado à palavra original. Os elementos de uma matriz *sempre* têm um tamanho fixo. Se algumas das palavras tiverem menos de 10 caracteres, o restante será

preenchido com espaços em branco. Se a resposta do aluno fosse guardada numa variável string comum (não dimensionada), a variável guardaria apenas a palavra sem os espaços em branco. Quando o programa testou se a palavra original e a do estudante eram iguais, elas não seriam, pois o computador considera os espaços em branco como parte da palavra.

Definindo o valor de C$ como 10, faz com que a resposta seja sempre preenchida em um tamanho-padrão de 10.

As duas variáveis definidas nas linhas 20 e 25 guardam o escore do usuário. RT guarda o número de respostas corretas e WR, o de erradas.

Essas linhas de inicialização são seguidas de um loop de entradas de dados que preenche o array com as palavras. Tão logo uma palavra "nula" seja lida, (isto é, quando você teclar **ENTER**) o programa entende que acabaram-se as entradas de dados. Você sempre pode entrar com um número de palavras inferior ao máximo. A palavra vazia é detectada na linha 60 que verifica se o valor digitado é menor do que um A.

Quando você usa qualquer um dos operadores relacionais (<, >, < =, >=, =, <>) com valores alfanuméricos (strings), o computador compara o código do primeiro caractere. A linha 60 do programa procura saber se o usuário digitou uma palavra feita de espaços em branco. Se isto acontecer o computador entende que o usuário não quer entrar com mais dados.

A segunda parte do programa é a seção operacional:

```
200 CLS
210 IF RT + WR> = 20 THEN GOTO 1000
220 LET W = INT ( + RND * T)
230 IF A$ (W) < "A" THEN GOTO 220
240 LET B$ = A$ (W)
250 FOR X = 1 TO R
260 LET N = INT (1 + RND * 10)
270 IF B$ (N) < "A" THEN GOTO 260
280 LET B$ (N) = "?"
290 NEXT X
300 PRINT B$
310 INPUT C$
320 PRINT C$
330 IF A$ (W) = C$ THEN GOTO 500
340 PRINT "ERROU. OUTRA TENTATIVA?"
350 INPUT C$
360 IF C$ (1) <> "N" THEN GOTO 300
370 LET WR = WR + 1
380 PRINT A$(W); "E A RESPOSTA"
390 LET A$(W) = " "
400 PAUSE 120
420 GOTO 200
```

A linha 210 decide se o exercício já se prolongou o bastante. É permitido tentar 20 palavras, estando elas certas ou erradas. Se você quiser que o programa dure mais, ou menos tempo, basta mudar o 20.

A linha 220 escolhe a palavra. Note que a variável T, a

que guarda o tamanho da array A$, é usada para determinar a faixa dos números randômicos.

A linha 230 trata do caso de palavras em branco na array. Se ela, escolhida randomicamente, for vazia (o que significa que o seu código é menor que o código de A) o programa escolhe outro número randômico.

A linha 240 transfere a palavra para outra variável. A cópia da palavra em B$ será alterada. A palavra original permanece intacta e pode ser usada outra vez.

O loop **FOR/NEXT** contido nas linhas 250/290 é o coração do programa. Essas linhas escolhem, randomicamente, posições de caracteres na palavra e colocam pontos de interrogação nessas posições. A linha 269 escolhe números entre 1 a 10. A linha 270 verifica se há uma letra neste espaço. Note-se que o espaço pode ser em branco, ter um ponto de interrogação ou um caractere. Ambos — espaço em branco e ponto de interrogação — têm códigos menores do que o código de A; logo, em qualquer caso o programa voltaria e escolheria outro número.

A linha 280 simplesmente coloca um ponto de interrogação na posição escolhida.

O loop se repete por R vezes. R é o número de caracteres a serem substituídos. Este número é escolhido pela pessoa que digitou a palavra a ser soletrada.

O exercício começa na linha 300, onde a palavra (com algumas letras substituídas por pontos de interrogação) é impressa na tela. O usuário digitou uma resposta que é comparada à palavra original. Se forem iguais, o usuário

está certo e é mostrada uma mensagem de parabéns através do comando **GOTO** 500; caso contrário, o programa diz que sua resposta está errada e convida o usuário a tentar de novo. Se o usuário digitar qualquer coisa que começa com N (o programa olha apenas o primeiro caractere — veja C$(1) na linha 360), soma-se 1 ao escore de erros e o programa soletra corretamente a palavra durante dois segundos (**PAUSE**120). Retira-se a palavra da lista e outra é escolhida.

Se o usuário digitar qualquer outra coisa em resposta à pergunta, o programa volta e imprime a mesma palavra de novo. Em todos os casos, as tentativas do usuário são impressas próximo à palavra-teste, e todas as tentativas são mostradas até que se escolha outra palavra.

A última seção do programa controla a mensagem de parabéns e também mostra o escore quando o exercício acaba. Aqui estão as linhas:

```
 500 FOR X = 1 TO 10
 510 LET N = INT (1 + RND*20)
 520 LET W = INT (1 + RND * 30)
 530 PRINT AT N, W; **VOCE ESTA CERTO**
 535 PAUSE 10
 540 PRINT AT N, W
 550 NEXT X
 560 LET RT = RT + 1
 570 GOTO 200
1000 PRINT "ACERTOS = "; RT, "ERROS="; WR
```

A tela dada quando o usuário responde corretamente está nas linhas 500 e 550. Neste loop, dois números são escolhidos: um entre 1 e 20 e outro entre 1 e 30. Eles são usados no comando **PRINT** nas locações **AT** (os valores 20 e 30 servem para manter a mensagem na tela. A tela tem 21 linhas e 31 colunas). Esses números são usados para escrever **VOCE ESTA CERTO** na tela. Imediatamente após ter sido escrita a mensagem, outro comando **PRINT AT** cobre a mensagem com caracteres em branco. Isto acontece 10 vezes, dando a impressão de que a mensagem pisca e passeia pela tela.

Uma vez terminada a mensagem, o escore do usuário é aumentado de 1, e o programa escolhe outra palavra.

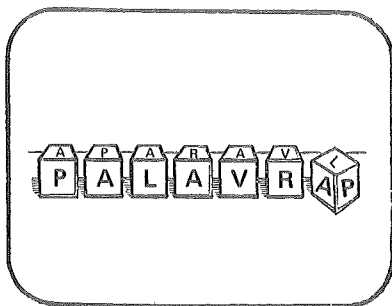
Quando o programa termina, ele vai para a linha 1000, imprime os escores e termina.

Se você quiser executar o programa de novo, sem colocar novas palavras, digite os seguintes comandos:

```
LET RT = 0
LET WR = 0
GOTO 200
```

Se o computador estiver no modo **FAST**, você deve colocá-lo no modo **SLOW**.

Usando **GOTO** 200 em vez de **RUN**, as variáveis permanecem intactas. Se você usasse **RUN**, o conteúdo de A$ seria destruído.

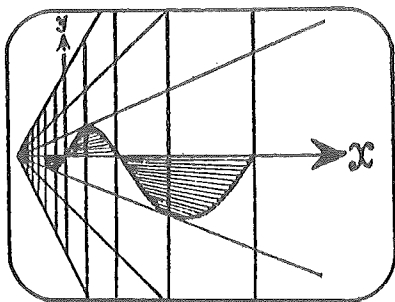
Anagrama

Este programa o ajudará a decifrar anagramas, isto é, um grupo de letras misturadas que formam uma palavra. Ele lê as letras digitadas e imprime todas as combinações possíveis até que se pare o programa. Eis o programa:

```
10 PRINT "PALAVRA?"
20 INPUT B$
30 LET L = LEN B$
40 LET N = 1 + INT (RND * (L - 1))
50 LET C$ = B$ (L)
60 LET B$ (L) = B$ (N)
70 LET B$ (N) = C$
80 PRINT B$ + "";
90 GOTO 30
```

Em resposta à pergunta "PALAVRA?", você digita um grupo de letras. Depois que você teclar **ENTER**, o pro-

grama guarda o número de caracteres da palavra (**LEN B$**) na variável L e gera um número entre 1 e o tamanho da palavra, menos 1. O programa, então, guarda o último caractere da palavra em C$, coloca o último caractere numa posição randomicamente escolhida, e coloca o caractere desta posição no final da palavra. Em resumo: o programa troca o último caractere da palavra por outra qualquer. Depois disso, ele modifica a palavra colocando um espaço no final dela, cria um espaço entre uma palavra e outra e troca os pedaços de posição. Parece confuso? Execute o programa algumas vezes e ele parecerá mais simples.

Este programa torna-se mais interessante quando a palavra é pequena. Por exemplo, quando há apenas 4 letras na palavra, há apenas 24 combinações possíveis. Com 5 letras há 120 combinações. Com 10 letras há 3628800 arranjos diferentes para as letras.

Como Traçar o Gráfico da Função Seno

A função **SIN** calcula o seno de um ângulo dado. Este ângulo deve estar em radianos. Você deve estar acostumado a medir ângulos em graus — há 360 graus em um círculo ou 2 π radianos. Aqui está o programa:

```
10 FOR X = 0 TO 2 * PI STEP 0.1
20 PLOT 10 * X, 20 + (10 * SIN X)
30 NEXT X
```

Um loop **FOR/NEXT** é definido por dois comandos separados: **FOR... TO... STEP** e **NEXT.**

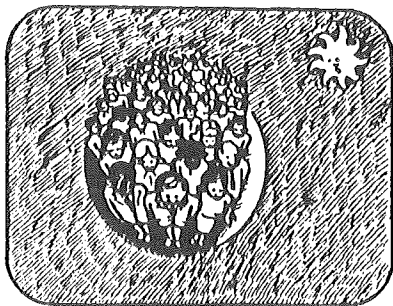
FOR inicia uma variável chamada contador do loop e lhe dá um valor inicial que é o primeiro após o sinal de igual. Na linha 10, o valor inicial é 0 (zero). O comando **FOR** inicia também um valor **TO** e **STEP.** Todos três valores podem ser expressões, como você vê na linha 10. As expressões são resolvidas ao executar-se o comando **FOR.**

Depois que o **FOR** é executado, o programa segue normalmente sua seqüência. Neste caso, o programa executa a linha 20 e prossegue até achar um comando **NEXT.** O valor de **STEP** é somado ao valor inicial do contador de loop e o computador verifica se o valor da variável do contador é maior que o valor final do loop. Se não for, o programa vai para o comando, imediatamente depois do **FOR.**

Este processo se repete até que o contador seja maior ou igual ao valor final do loop. O loop acaba, e após o

NEXT, é executada a próxima instrução. No exemplo anterior, terminando o loop **FOR/NEXT** o programa acaba.

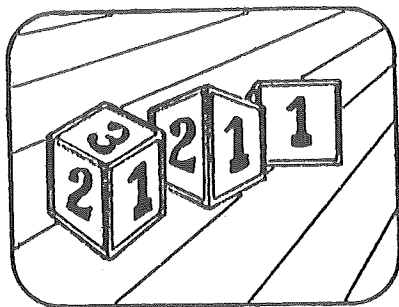
Se você não usar **STEP**, considera-se o incremento como 1. Se o valor de **STEP** for negativo, o valor de **TO** tem que ser *menor* do que o valor inicial. Quando o valor de **STEP** for positivo e o valor inicial for maior ou igual ao valor de **TO**, o loop será ignorado.

Crescimento Populacional

Este programa é um exercício de cálculos. De acordo com pesquisas, a população crescerá a uma taxa de 1.8% ao ano nos próximos vinte anos.

O programa calcula a população futura, estimando a população atual em 4.5 bilhões e a taxa de 1.8% ao ano.

Este programa prossegue indefinidamente. Quando você se cansar, tecle **BREAK**:

```
10 LET B = 4500000000
20 LET Y = 1982
30 LET Y = Y+1
40 LET B = B+.018 * B
50 IF  Y/100  =  INT  (Y/100)  THEN  PRINT
   Y,B/1000000000; "BILHÕES
60 GOTO 30
```

Fatoriais:

Uma das ferramentas básicas da probabilidade é o fatorial. Em uma série de eventos interligados, a probabilidade que cada evento tem de ocorrer é dada pelo fatorial do número total de elementos. O fatorial é o produto de todos os números usados para contar o número em questão. Por exemplo: o fatorial de 10 seria 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 ou 3628800.

Por exemplo, suponhamos que você queira saber quantas combinações das letras M, E e U existem. Se você for colocar as letras, há três posições que a primeira pode ocupar, duas para a segunda e uma para a terceira. Logo, há 3 * 2 * 1 ou 6 combinações possíveis. Aqui estão as combinações:

MEU MUE UEM UME EUM EMU

Este programa acha o fatorial de um número:

```
 5 PRINT "DIGITE UM NUMERO"
10 INPUT A
20 FOX X = A 1 TO 2 STEP 1
30 LET A = A * X
40 NEXT X
50 PRINT A
```

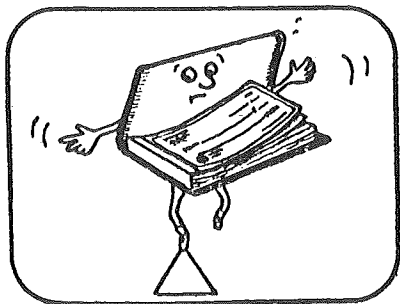
O loop **FOR/NEXT**, que começa na linha 20, usa como contador a variável X que vai de uma unidade a menos que o número, até 2. Um loop **FOR/NEXT** pode variar de qualquer valor e até frações. Basta incluir um **STEP**. X começa em a-1 pois é sempre multiplicado por A e vai até 2, já que não é necessário multiplicar o número por 1.

A variável A guarda o total. Uma vez lido seu valor inicial, o loop não precisa iniciar com seu valor.

Após o processamento a resposta é impressa na tela.

10

PROGRAMAS CASEIROS

Balanço do Talão de Cheques

Este programa o ajudará a fazer o balanço do seu talão de cheques.

Como falamos antes, quando você decide fazer um programa, deve planejar primeiro o que o programa deve fazer. Então, é hora de escolher os comandos, em **BASIC**, necessários para a tarefa.

A primeira informação que o seu computador precisa saber é o início do seu balanço. A seguir, você vai entrar com o valor de cada cheque ou depósito. Finalmente, o computador deve calcular e mostrar o saldo atualizado.

Eis uma lista daquilo que o programa deve fazer:

1. Ler o saldo anterior.
2. Ler o valor de um cheque.
3. Há mais cheques? Se houver, volte para 2.
4. Se não houver mais cheques, imprima novo saldo.
5. Finalizar.

Você já viu as instruções **PRINT** e **INPUT**. Usando **INPUT**, seu programa lê os dados do usuário para o balanço. Entretanto, se você usar apenas um comando **INPUT**, o usuário não saberá qual o dado que o computador espera. Logo, é necessário usar o comando **PRINT** antes de um comando **INPUT**, para dizer ao usuário o que fazer.

```
100 PRINT "BALANÇO"
150 INPUT BAL
```

BAL é a variável usada para guardar os valores do saldo. Você poderia chamar a variável de BALANÇO ou SALDO, porém nomes maiores ocupam maior espaço na

memória. Você descobrirá que o espaço na memória é um grande problema e, então, quererá minimizar os seus programas o máximo possível; você sempre terá que sacrificar certo grau de clareza para ganhar um espaço extra na memória. Você poderia mandar o computador imprimir mensagens maiores como "POR FAVOR, DIGITE O SALDO ANTERIOR SEM O CIFRÃO", mas isto usaria mais espaço que o necessário. O uso de mensagens curtas é perfeitamente aceitável, especialmente se você for o principal usuário de seus programas.

Agora você precisa entrar com o valor do primeiro cheque. Tente digitar estas três linhas:

```
200 PRINT "VALOR"
250 INPUT VA
300 LET BAL = BAL-VA
```

Após a linha 300, BAL guarda o novo saldo.

O terceiro passo do plano requer que o programa descubra se há mais cheques para serem lidos. Devemos estabelecer uma regra, para que quando o usuário digitar o valor 0, indicando que não há mais cheques, o programa termine; isto termina com o terceiro passo.

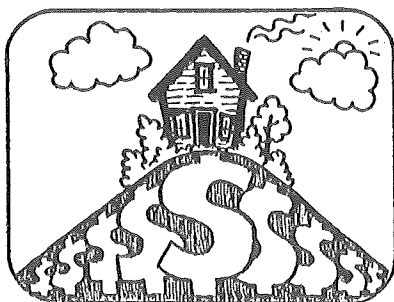
```
350 IF VA = 0 THEN GOTO 500
400 GOTO 200
500 PRINT "SALDO ATUAL: CR$"; BAL
```

Qualquer número de itens pode aparecer em um comando **PRINT**. Cada um deles deve ser separado por uma

vírgula [,] ou por um ponto-e-vírgula; o valor da variável BAL é impresso imediatamente após, na mesma linha.

Eis o programa:

```
100 PRINT "BALANÇO"
150 INPUT BAL
200 PRINT "VALOR (DIGITE 0 PARA FINALIZAR)"
250 INPUT VA
300 LET BAL=BAL-VA
350 IF-VA = 0 THEN GOTO 500
400 GOTO-200
500 PRINT "SALDO ATUAL: CR$"; BAL
```

Compra de uma Casa

Descobrir o valor das prestações de um imóvel pode ser doloroso, com as atuais taxas de juros. Suponha que a

casa de seus sonhos custe 50.000 dólares a uma taxa de juros de 20%. Parece claro, não? Qual o próximo passo? Nós, por nossa vez, não conseguimos entender como os bancos chegam a essas mensalidades para pagamentos em trinta anos. Você já pode adivinhar o propósito deste programa (ele também pode ser usado para saber as prestações de seu automóvel).

Para descobrir os pagamentos mensais da casa, basta digitar os valores quando o programa perguntar o PREÇO DA CASA. O valor total da casa pode deixá-lo chocado.

Eis o programa:[13]

```
 10 PRINT "PREÇO DA CASA?"
 20 INPUT P
 30 PRINT CR$"; P
 40 PRINT "NÚMERO DE ANOS?"
 50 INPUT A
 60 PRINT A
 70 PRINT "TAXA DE JUROS?"
 80 INPUT J
 90 PRINT J; "POR CENTO"
100 LET J = J/100
110 LET PMT = P * J/((1+J)*(1-(1+J) **-A))
120 PRINT "PAGAMENTO MENSAL: CR$"; PMT/12
130 PRINT "TOTAL PAGO: CR$"; PMT * A
```

---

[13] O programa aplica-se ao sistema de financiamentos dos EUA.

Planos de Poupança

Este programa também pode ser chamado "Planos para se Aposentar" ou "A Verdade sobre sua Caderneta de Poupança". Ele computa seu saldo usando seus depósitos mensais, taxas de juros, número de anos e, é claro, a inflação.

Aqui está o programa:[14]

```
10 LET T = 0
15 LET D = 0
20 PRINT "DEPOSITOS MENSAIS?";
25 INPUT M
30 PRINT "CR$"; M
35 PRINT "NUMERO DE ANOS?";
40 INPUT A
45 PRINT A
```

[14] O programa está de acordo com o sistema de poupança dos EUA.

```
 50 PRINT "TAXA DE JUROS?";
 55 INPUT I
 60 PRINT I; "POR CENTO"
 65 PRINT "INFLAÇÃO?";
 70 INPUT INF
 75 PRINT INF; "POR CENTO"
 80 FOR X = 1 TO A
 85 LET AAD = 0
 90 FOR Z = 1 TO 12
 95 LET AD = M-T * I/100/12
100 LET T = T + AD
105 LET AAD = AAD + D
110 NEXT Z
115 LET D = D + AAD - (D+AAD) * INF/100
120 NEXT X
125 PRINT "TOTAL ECONOMIZADO: CR$:"; T
130 PRINT "ACIMA DA INFLAÇÃO"
```

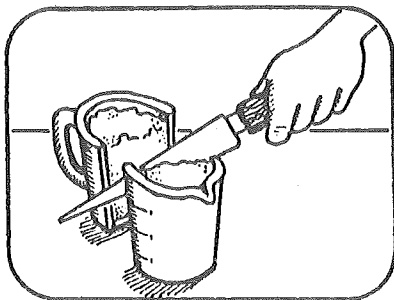
Para usar este programa, apenas digite **RUN** e responda às perguntas. Os números por você digitados são repetidos para que você veja com o que começou, e o resultado final. A repetição é feita com comandos **PRINT** nas linhas 30, 45, 60 e 75. Observe que os números estão nas mesmas linhas que as perguntas; isto se obtém colocando-se ponto-e-vírgula [;] ao final das linhas do comando **PRINT** onde estão as perguntas.

O programa usa a taxa de inflação dada, para calcular o valor real do dinheiro. A taxa de juros é avaliada mensalmente, uma decisão arbitrária. Se você quiser uma preci-

são maior, altere o programa trocando os 12's nas linhas 90 e 95 por outros números (365 para composição diária, 4 para composição quaternária) ou por uma variável. Se você usar uma variável, terá que incluir outras seqüências de **PRINT**'s e **INPUT**'s para a nova variável. Se você quiser saber com mais detalhes como a composição é usada, veja o programa para calcular o retorno de um investimento, no próximo capítulo.

A inflação é composta apenas uma vez ao ano, já que esta é a maneira como os números da inflação são publicados. A diferença na composição dos períodos torna o segundo loop necessário. Se a inflação fosse também calculada mensalmente o programa necessitaria de apenas um loop **FOR/NEXT**.

Este programa leva algum tempo para processar. Pode-se aumentar sua velocidade executando-o no modo **FAST**.

Conversão de Medidas

Todos nós já tivemos um professor de física, que insistia em nos dar problemas sobre medidas. Há algo em nós que aprova o sistema americano de medidas! O pé era a medida do pé do rei; o acre era a porção de terra que a pessoa podia arar em um dia.

Romance é uma coisa, mas conversão de medidas é outra. Nunca conseguimos nos lembrar quantas colheres de sopa há em uma xícara ou quantas colheres de chá em uma onça.

As coisas se complicam quando você precisa converter uma receita para alimentar mais, ou menos pessoas. Nosso livro de receitas favorito recomenda usar quantidades que dificilmente podemos calcular. Se você tem este problema, este programa lhe é destinado.

O programa pergunta primeiro o número de medidas na receita original e, depois, o número de medidas que você quer. Se está simplemente convertendo entre instrumentos de medidas (por exemplo, quer saber quantas colheres de chá há em uma onça), digite o mesmo número para o número original de medidas e o número desejado. O programa pergunta, então, que quantidade você quer converter e a qual unidade. O programa lê, apenas, os dois primeiros caracteres que você digita (CS - colher de sopa, CC - colher de chá, ON - onça e XI - xícara). Se ele não reconhecer sua entrada, simplesmente perguntará de novo.

A seguir, ele converte as medidas, mostrando as novas quantidades em cada uma das unidades. Você é pergun-

tado se há mais conversões. Se a resposta for sim, o programa pede a outra quantidade para converter.

Eis o programa:

```
  1 PRINT AT 20,2"***CONVERSÃO DE RECEITAS***"
  2 PAUSE 100
  3 FOR X = 1 TO 20
  4 SCROLL
  5 NEXT X
  6 CLS
 10 DIM Q$(2)
100 PRINT "NUMERO ORIGINAL DE MEDIDAS?"
110 INPUT S
120 PRINT "NUMERO QUE VOCE DESEJA?"
130 INPUT N
140 LET N = N/S
150 CLS
160 PRINT "QUANTIDADE ANTERIOR:"
170 INPUT Q
180 PRINT "UNIDADES:"
190 INPUT Q$
195 CLS
200 IF Q$ <>"CC" AND Q$<>"CS" AND Q$<>
    "ON" AND Q$<> "XI" THEN GOTO 180
210 IF Q$ = "XI" THEN LET Q = Q * 48
220 IF Q$ = "ON" THEN LET Q = * 6
230 IF Q$ = "CS" THEN LET Q = Q * 3
240 LET Q = Q * N
```

```
250 PRINT INT (100 * Q)/100; "COLHERES DE CHÁ",
    INT (100 * Q/3)/100;
260 PRINT "COLHERES DE SOPA", INT (100 *
    Q/6)/100; "ONÇAS"
270 PRINT INT(100 * Q/48)/100; "XICARAS"; AT 20,1;
    "MAIS CONVERSÕES?"
280 INPUT Q$
290 IF Q$ <>"NÃO" THEN GOTO 150
```

As linhas de 1 a 6 são um começo decorativo para o programa. O comando **PAUSE** suspende a execução do programa. Há 60 unidades de pausa em um segundo; então, **PAUSE** 100 espera por 1.66 segundos. **SCROLL** apaga a linha superior da tela e move todo o resto, uma linha para cima. O comando **CLS** na linha 6 é necessário porque, após o comando **SCROLL**, o próximo comando **PRINT** aparece na última linha da tela.

O programa guarda o número original de medidas na variável S, e o novo número de medidas na variável N dividindo, então, N por S para obter o fator de conversão. Note que o fator de conversão é guardado em N para evitar o uso de uma variável extra.

A linha 200 verifica se a unidade lida é aceitável pelo programa.

O tamanho de Q$ é definido com 2 em um comando **DIM**, na linha 10. Quando o tamanho de uma variável string (alfanumérica) é definido em um comando **DIM**, esta variável string sempre terá este tamanho; sendo assim, todos os caracteres lidos depois do dois primeiros

são ignorados. Por exemplo, se xícara fosse digitado, Q$ teria o valor de XI, que é um dos valores aceitos. Se o usuário digitasse COS em vez de CS, o programa não aceitaria esta entrada e repetiria a pergunta.

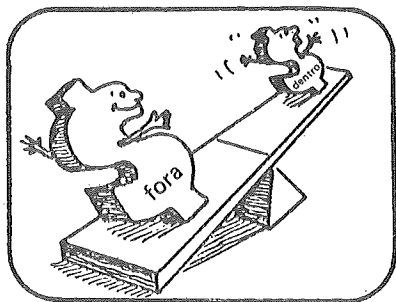
Cada um dos comandos **PRINT** nas linhas 250 a 258 inclui uma expressão **INT** (100 * Q)/100. Q contém o número convertido, de colheres de chá, perguntado. O resto desta expressão são apenas dois caracteres à direita do ponto decimal e que serão mostrados. Uma quantidade de 4.3567123 xícaras não ajudaria muito, já que a precisão necessária para cozinhar é 4.35.

Para simplificar as coisas, todas as quantidades foram convertidas em colheres de chá, antes de serem processadas. Pelo mesmo motivo, a nova quantidade é mostrada nas quatro unidades. É possível, também, escrever um programa que decida qual a unidade mais adequada.

Outro programa útil seria um que convertesse medidas inglesas para o sistema métrico decimal. Pode parecer difícil, mas é bem parecido com o programa acima. Você pode até tentar combinar ambos em um só programa.

11

PROGRAMAS DE NEGÓCIOS

Balanço Mensal

Fazer seu balanço mensal pode ser complicado, especialmente se houver despesas que não são mensais (como quotas de alguns clubes, por exemplo).

Este programa é muito flexível e facilmente expandível. Pode, também, conter fórmulas para conversão de despesas ocasionais em despesas mensais. Com a memória de seu T/S 1000, você pode expandir este programa para incluir tudo o que quiser.

Aqui está o programa:

```
 10 DIM E(5)
 15 DIM M$(5,5)
 20 DIM F$(5,8)
 25 LET M$(1) = "ALUG"
 30 LET F$(1) = "E (A)"
 35 LET M$(2) = "CLUB"
 40 LET F$(2) = "E(A)/2"
 45 LET M$(3) = "ROUPA"
 50 LET F$(3) = F$(1)
 55 LET M$(4) = "AM/SE"
 60 LET F$(4) = "E(A) * 4.3"
 65 LET M$(5) = "OUTRO"
 70 LET F$(5) = F$(1)
100 PRINT "RECEITA"
110 INPUT I
130 FOR A = 1 TO 5
140 PRINT M$(A)
150 INPUT E$(A)
160 CLS
```

```
170 LET E(A) = VAL F$(A)
180 LET I = I-E(A)
190 NEXT A
200 FOR A = I TO 5
210 PRINT M$(A); "= CR$"; E(A);"/MES"
220 NEXT A
230 PRINT "EXTRA = CR$"; I
```

O segredo deste programa está em se compreender a função **VAL**. Ela retorna o valor numérico de uma string. A variável alfanumérica indexada F$ contém fórmulas para converter os números lidos.

Por exemplo: considere a linha 40. F$(2) contém a string E(A)/2. E(A)/2 é o número lido que representa o dinheiro pago pela taxa de um clube. Suponha esta taxa bimestral. Logo, a fórmula F$(2) divide a taxa bimestral convertendo-a em sua mensal equivalente. Por outro lado, você calcula seus gastos com alimentação por semana. Para converter isto em um gasto mensal, F$(4) multiplica E(A) por 4.3 já que há, aproximadamente, 4.3 semanas em um mês.

O loop começa na linha 130 mostra um elemento da array M$ que contém mensagens identificando as despesas, lê os valores digitados e os coloca na array E. A linha 170 converte o valor, usando a fórmula da array F$. Quando o elemento é processado em F$ usando-se **VAL**, tudo se passa como se estes valores estivessem nesta linha. Por exemplo: na segunda vez que o loop for processado, **VAL** F$ (A) será igual a E (A)/2.

Este programa poderia ser escrito sem loop's **FOR**. Bastava escrever separadamente os itens, usando **PRINT**, **INPUT** e uma linha de conversão para cada categoria. Porém, escrever o programa usando **VAL**, arrays e um loop **FOR/NEXT**, economiza espaço.

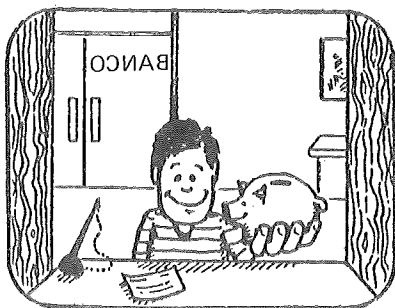
Se você quiser acrescentar mais itens da despesa, tudo o que precisa fazer é aumentar o tamanho das arrays e, então, colocar as mensagens apropriadas nos elementos das arrays. Você terá que alterar os valores **TO** nos comandos **FOR**.

Por exemplo: acrescente as seguintes linhas para criar uma categoria para gastos com o automóvel (algumas destas linhas substituem as linhas do programa original):

```
 10 DIM E(6)
 15 DIM M$(6,5)
 20 DIM F$(6,8)
 80 LET M$(6) = AUTO
 90 LET F$(6) = F$(1)
130 FOR A = 1 TO 6
200 FOR A = 1 TO 6
```

Naturalmente você deve ter outras despesas que foram esquecidas. Talvez queira incluir gastos com telefone, lavanderia, diversões e outros. Até sem expansão de memória para seu computador não será problemático acrescentar alguns itens. Os elementos de array M$, podem ser maiores, tornando as mensagens mais claras.

Mantivemos as mensagens curtas para fazer o programa menor e aumentar o espaço-memória.

Cálculo do Retorno de um Investimento

Quando o nosso banco começa a compor os juros para nossos depósitos, paramos para pensar: quanto teríamos, em cinco anos? Quem sabe?

Este programa começa com um certo capital, uma taxa fixa de juros e uma taxa de composição fixa e, então, calcula o valor do capital inicial a cada ano.

Eis o programa:

```
100 PRINT "CAPITAL?"
110 INPUT A
120 PRINT "NUMEROS DE ANOS?"
130 INPUT Y
140 PRINT "TAXA DE JUROS?"
150 INPUT I
```

```
160 PRINT "NUMERO DE COMPOSIÇÕES"
170 PRINT "POR ANO"
180 INPUT C
190 CLS
200 FOR X = 1 TO Y
210 FOR Z = 1 TO C
220 LET A = A + A * I/C/100
230 NEXT Z
240 PRINT "ANO"; X, "CR$" A
250 NEXT X
```

Nas linhas 200 e 210 começam 2 loops **FOR/NEXT**. Eles são chamados loop's embutidos (nested), isto é, um está completamente incluído no outro. O loop externo representa os anos e o interno o número de vezes que a taxa de juros é composta, ao ano.

Se um loop começa depois do outro, o segundo loop deve estar contido no primeiro (nested). Em outras palavras: se **FOR-X** está antes de **FOR-Z**, **NEXT-Z** deve vir antes de **NEXT-X**. Uma vez iniciado o loop interno, ele deve completar todo o seu ciclo antes de procurar o **NEXT** do loop externo. Se o computador chegar ao **NEXT** do loop externo antes do **NEXT** do loop interno, o programa pode se perder, isto é, "perder o fio da meada." De qualquer forma, seu programa não deve fazer isto. Logo, verifique se seus loop's são independentes ou estão propriamente embutidos.

Outra linha interessante é a 220:

```
220 LET A = A + A * I/C/100
```

Esta fórmula deve calcular o novo saldo após somados os juros. A parte que representa os juros é I/C/100. Isto quer dizer que você digita os juros como se fossem uma porcentagem. Por exemplo: quando o computador pergunta "TAXA DE JUROS?" e se a taxa for 17%, você deve digitar 17. Juros são a porcentagem somada a seu capital, após cada ano. Porcentagem é um número tirado de 100. Por exemplo: para se obter 17% de Cr$ 1.000, basta multiplicar Cr$ 1.000 por 17 e dividir por 100. A variável C está na fórmula I/C/100 porque este é o número de vezes que a taxa é composta ao ano. Quando um banco compõe seus juros, calcula quanto rendeu seu dinheiro naquele período e soma ao saldo anterior. Se a taxa for de 17% uma vez ao ano, um depósito de Cr$ 1000 renderá Cr$ 170 perfazendo Cr$ 1.170. Se a taxa for composta a cada três meses, Cr$ 42,50 serão somados no final do 1.º trimestre, Cr$ 44,31 no 2.º, Cr$ 46,19 no 3.º e Cr$ 48,15 ao final do último trimestre, perfazendo Cr$ 181,15. A diferença existe, pois quando foi feito o cálculo dos juros do 2.º trimestre, os juros do 1.º trimestre já haviam sido somados ao saldo. O mesmo acontece com os 3.º e 4.º trimestres.

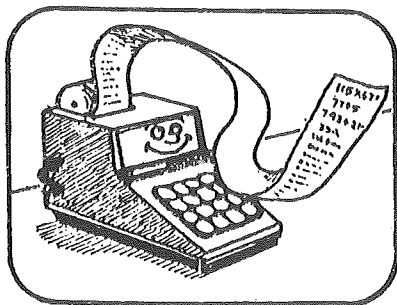
Para usar este programa basta digitar o capital inicial, o número de anos desejado, a taxa de juros (sem o sinal de porcentagem: se a taxa for 5%, digite 5; se for 5 1/4%, digite 5.25) e o número de vezes que a taxa é composta ao ano. O programa irá listar o valor de seu investimento ao final de cada ano. Se a taxa for composta várias vezes ao ano, o programa se tornará lento. Execute-o no modo **FAST**. Se você usar um período superior a 22 anos, o pro-

grama pára pois a tela ficará cheia. Se isto acontecer, basta digitar **CONT** para limpar a tela e acompanhar os demais resultados.

Eis um exercício para este programa:

Suponha que o Banco Central leve dois dias para creditar um cheque. Um banco de grande porte leva, por hipótese, seis dias para creditar na conta do cliente. Ou seja: O banco fica com este dinheiro por 4 dias e, é claro, aplica-o. Suponha que ele empreste este dinheiro a uma taxa de 16%. Suponha, também, que o banco tenha um milhão de clientes e cada um espera um cheque de Cr$ 100 a ser creditado em sua conta. Isto deixa o banco com Cr$ 100.000.000, pelo período de um ano, a uma taxa de 16%, composta diariamente (não digite % ou Cr$).

Seu programa deve dar como resposta um lucro de, aproximadamente, Cr$ 17.000.000.

Uma Calculadora Melhorada

Este programa é extremamente simples, embora funcione melhor que uma calculadora (será?). Eis o programa:

```
10 INPUT A$
20 PRINT A$; " = "; VAL A$
30 GOTO 10
```

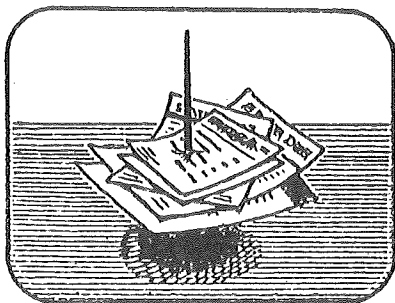
Quando você executar este programa verá um par de aspas na parte inferior da tela. Tudo o que você precisa fazer é digitar a expressão matemática e teclar **ENTER.** O computador mostrará sua equação e a resposta. A expressão pode ser tão grande quanto se queira, mas use apenas constantes, funções e operadores. Você pode usar, por exemplo, as seguintes expressões:

```
SIN 1
2 ** 3 - 8 * 7
3 ** RND
```

A função **VAL** chama o valor da expressão para resolver a fórmula digitada.

A grande vantagem que este programa tem sobre uma calculadora, é a capacidade de mostrar a fórmula inicial; logo, você pode verificar se houve algum erro (numa calculadora isto não é possível).

O programa termina quando uma entrada igual é lida pelo computador; uma letra, por exemplo.

Bloco de Anotações

Este programa guarda anotações, listas de contas pagas ou qualquer coisa que você desejar.

É estruturado de maneira simples, de modo a reservar o maior espaço possível para suas anotações.

Ele permite três operações: pode adicionar anotações, rever as anotações já salvas ou salvar-se em fita cassete. Cada função está em uma seção diferente.

Aqui está o programa. Não tente executá-lo antes que você acabe de ler tudo a respeito, pois ele não funcionará sem uma certa "inicialização".

```
10 PRINT "[A]DICIONAR,[R]EVER,[S]ALVAR"
20 INPUT B$
30 CLS
40 GOTO 100 * CODE B$
3800 PRINT "ANOTAÇÃO"
```

```
3810 INPUT B$
3820 LET A$ = A$ + B$ + "**"
3830 GOTO 10
5500 FOR X = 1 TO LEN A$
5510 IF A$(X) <> "**" THEN PRINT A$(X);
5520 IF A$(X) = "**" THEN PRINT
5530 NEXT X
5540 GOTO 10
5600 SAVE "RASCUNHO"
5610 GOTO 10
```

A linha 10 mostra o "menu" das operações possíveis. A linha 20 lê a opção (você pode digitar a palavra inteira ou a primeira letra, apenas).

A linha 30 limpa a tela. Isto evita confusões e economiza espaço na memória. Baseada em sua escolha, a expressão 100 * **CODE** — B$ é uma maneira compacta de dizer que rotina foi escolhida. Cada caractere tem um código interno diferente e a função **CODE** retorna o código de um caractere lido. O código do caractere A é 38, de R é 55 e S é 56. Logo, o programa irá para as linhas 3800, 5500 e 5600 dependendo da escolha. Observe que o programa não distingue entradas ilegais. Se você digitar qualquer coisa que não comece por A, R ou S, o programa não fará o que você desejar.

A seção que trata de adicionar anotações, vai da linha 3800 a 3830. Ela pergunta pela anotação e depois guarda-a na variável B$. A fim de economizar espaço, a variável do "menu" é reutilizada. O programa, então, pega o

antigo valor de A$, adiciona as novas anotações em seu final, e adiciona, também, dois asteriscos (**). Os asteriscos podem ser digitados usando-se o sinal de multiplicação. A$ passa a conter todas as antigas anotações, somadas às novas com ** no final. Os ** são usados para distinguir as anotações, e são também usados pela próxima seção do programa.

A seção para rever as anotações vai da linha 5500 a 5540. Ela usa um loop **FOR/NEXT** para percorrer A$ da 1.ª posição até o final (**LEN**-A$ dá o tamanho total de A$). A linha 5510 testa cada caractere para verificar se é um duplo asterisco. Se não for, a letra é impressa. Note que a linha termina com um ponto-e-vírgula [;]. Isto diz ao computador para imprimir o próximo item na mesma linha. A linha 5.520 também verifica se o caractere é um asterisco. Se for, um comando **PRINT** é executado, de forma que o próximo caractere é impresso no início da próxima linha. Isto faz com que cada anotação fique em uma linha. Quando um duplo asterisco foi lido, significa que a anotação acabou. Os asteriscos, é claro, não são impressos.

A última seção do programa, linhas 5600 e 5610, é chamada se você digitar **SALVAR** no menu. A linha 5600 salva o programa em fita cassete; para fazer isto, prepare seu gravador, deixe-o funcionando, digite S e tecle **ENTER** em respota ao menu. Uma vez salvo o programa, enquanto ele estiver em execução tire a fita cassete e ele voltará imediatamente à execução. A linha 5610 é executada e o programa volta para o menu (ver Capítulo 5). Após a execução da instrução **SAVE** o programa volta ao me-

nu. Você continua a usá-lo, ou sai do programa. Para sair, **DELETE** as aspas à esquerda, na parte inferior da tela, digite **STOP** e tecle **ENTER.** Você pode, simplesmente, digitar, T, U, V, W, X, Y ou Z. O comando **GOTO** fará com que o programa vá para uma linha inexistente e o programa parará.

Se você tentou executar este programa, terá descoberto que ele não funcionou. Se tentar adicionar uma anotação, o programa falhará na linha 3820 porque A$ está no lado esquerdo de um comando **LET** e ainda não foi definido. Para executar o programa, digite os seguintes comandos (note que não há números nas linhas):

LET A$ = " "
GOTO 10

(As duas aspas são obtidas digitando-se duas vezes **SHIFT**/P e, não, digitando-se **SHIFT**/Q).

Nunca use **RUN** para executar este programa, pois isto vai zerar todas as variáveis e apagar suas anotações. Use **LET**-A$ = " " apenas a primeira vez que você for salvar anotações em seu programa. Se usar **RUN** após salvar as anotações, estas serão apagadas. Se você salvar o programa com a opção S do menu, nunca terá que iniciar sua execução.

É uma boa idéia usar uma cópia diferente do programa para diferentes tipos de anotações. Uma vez que a memória do computador é limitada, você não quererá "espremer" tudo em uma só cópia do programa. Além do mais,

será mais fácil achar informações específicas se você adquirir as informações por tipo. Para fazer uma nova cópia do programa, carregue uma cópia velha da fita cassete para o programa e digite os comandos **LET** e **GOTO**, acima. Você, agora, estará começando com uma cópia vazia de A$.

Este programa, da maneira que está, tem um grande inconveniente: você não pode remover qualquer anotação. Para remediar esta situação, digite as seguintes linhas:

```
5521 IF A$ = "**" THEN INPUT B$
5522 IF B$ <> "REMOVER" THEN GOTO 5530
5523 LET B$ = A$ (X + 1 TO)
5524 LET X = X-1
5525 IF A$ (X) <>"**" THEN GOTO 5524
5526 LET A$ = A$ (TO X) + B$
5527 GOTO 10
```

Quando você usar a função REVER com a rotina acima no programa, ele espera por uma entrada alfabética após cada anotação que for impressa. Se você teclar **ENTER** (ou qualquer coisa que não seja REMOVER) a próxima anotação será impressa. Se você, entretanto, digitar REMOVER, o programa guarda o resto da string B$ (uma vez mais B$ é usada para evitar variáveis extras). Observe o comando usado: A$ (X + 1 **TO**). A Xésima posição contém o duplo asterisco, logo * + 1 apontará para o início da

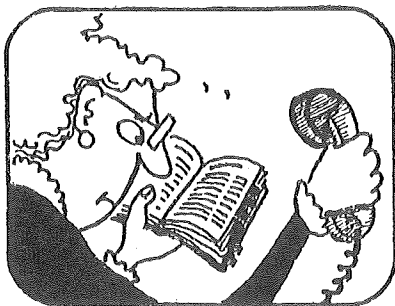
próxima anotação. A frase X + 1 **TO** refere-se a toda string, do início da próxima anotação até o fim.

Uma vez que a informação está guardada em B$, as linhas 5524 e 5525 voltam atrás em A$, para procurar o duplo asterisco, imediatamente em frente da anotação não desejada. Quando a anotação é encontrada, a linha 5526 muda A$ para conter as anotações desde o início até o duplo asterisco em frente da anotação deletada, (A$ (**TO** X)) e o conteúdo de B$. Já que B$ contém tudo de A$, que estava depois da anotação deletada, a anotação deletada é "cortada" do A$, deixando todo o resto intacto.

A linha 5527 retorna ao menu. Você bem imagina por que a rotina de exclusão não permite que o loop de revisão continue. Uma vez que a anotação foi retirada, A$ agora é menor; e, contudo, o loop **FOR/NEXT** foi estabelecido pelo comprimento original A$ e não pode ser alterada sem que se inicie o loop de novo. Seria possível guardar o valor de A$ no momento da interrupção e, então, reiniciar o loop com o novo tamanho de A$, mas isto requer uma ou duas variáveis e várias linhas de programa e isto tudo tomaria mais espaço na memória. Se você comprar uma expansão de memória, talvez queira fazer estas mudanças. Já que o espaço é necessário para guardar informações adicionais, é melhor voltar ao menu e começar tudo de novo.

Este programa ilustra uma maneira geral de salvar anotações. Você poderia salvar cada anotação em um elemento de uma array alfanumérica, mas isto tomaria mais espaço na memória, pois os elementos de uma array

devem ter um tamanho-padrão. As informações ocupam o mínimo de memória, se guardadas em strings. Se separadas por um caractere-padrão, o computador pode dizer quando cada informação termina. Este método (conhecido como *gravação indexada de tamanho variável*) é similar aos métodos usados por sofisticados sistemas de computação para salvar dados.

Caderno de Telefones

Não é um grande "programa", mas não é um caderno de telefones tão ruim.

O programa é composto de tantos grupos das próximas duas linhas quantos forem necessários.

```
5 PRINT "NOME", "TELEFONE", "ENDEREÇO"
10 STOP
```

Seu caderno de telefones pode ser composto de tantos grupos quantos forem necessários. Por exemplo:

```
 5 PRINT "TIA JULIA", "556-6600", "RIBEIRA"
10 STOP
15 PRINT "MUSEU HISTORICO", "558-2993". "PARQUE
   DA CIDADE"
20 STOP
25 PRINT "PREFEITO", "558-3456", "PREFEITURA"
30 STOP
```

Esta lista é acessada por comandos **GOTO.** Você não precisa decorar todos esses números; por quanto tempo consegue memorizar que 5 é Tia Júlia, e etc.? Use comandos **LET** para guardar o número das linhas em variáveis fácèis de serem lembradas. Por exemplo:

```
LET JULIA = 5
LET MUSEU = 15
LET PREFEITO = 25
```

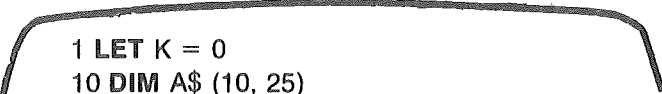
Você pode guardar o número de linhas em mais de uma variável, se tiver dificuldade de lembrar.

Para usar esta lista, tudo o que você precisa fazer é digitar um comando como este:

```
GOTO MUSEU
```

O computador imprime o conteúdo do comando **PRINT** na linha 15 e pára.

Se você salvar este programa em fita cassete, todas as variáveis também serão salvas. Lembre-se de nunca digitar **RUN** e **CLEAR** quando o programa estiver na memória

do computador, do contrário você apagará as variáveis e terá que recarregar o programa.

Este programa tem a desvantagem de cada linha comportar apenas um número de telefone; se você tiver dois amigos com o mesmo nome, terá que usar duas variáveis-nome diferentes para cada um. Este é um problema para quem tem dificuldades de lembrar nomes e organizar seu caderno de telefones. Para o primeiro nome na página, J, há dezenas de JOÕES ou JOSÉS. Este problema pode ser resolvido fazendo-se o computador imprimir todos os nomes de uma dada letra. Assim, todos os nomes iniciados por J seriam mostrados e você pegaria o que estivesse procurando. Neste caso, não teria que pôr em ordem alfabética todos os nomes.

O seguinte programa de caderno de telefones é muito mais complexo, porém mais eficiente.

O programa apresentado tem quatro funções: você pode adicionar nomes em seu caderno, alterar dados, ver os telefones e salvar o programa em fita cassete.

O programa se salva em fita cassete, enquanto está em execução. Quando você o carrega da fita, de novo, ele continua em execução. Se parar o programa, use um **GOTO** 30 para reiniciá-lo ou você destruirá todos os dados.

Antes de discutirmos como o programa está estruturado, aqui está ele:

```
1 LET K = 0
10 DIM A$ (10, 25)
```

```
  20 DIM B$ (1)
  30 PRINT "ADICIONAR, MODIFICAR, REVER,
          SALVAR"
  40 INPUT B$
  50 CLS
  60 GOTO CODE-B$ * 100
3800 LET K = K + 1
3820 IF K > 10 THEN GOTO-30
3830 LET E = K
3840 GOTO 6000
4000 PRINT "ENTRE COM NUMERO"
4010 INPUT E
4020 IF E > 0 AND E<11 THEN FOTO 6000
4030 GOTO 30
5500 PRINT "LETRA"
5510 INPUT B$
5520 FOR X = 1 TO K
5530 IF A$ (X, 1) = B$ THEN PRINT X; "   ";A$ (X)
5540 NEXT X
5550 GOTO 30
5600 SAVE "ENDEREÇOS"
5610 GOTO 30
6000 PRINT "NOVA ENTRADA DE DADOS"
6010 INPUT A$ (E)
6020 GOTO 30
```

Não é um programa muito longo, mas faz muita coisa. Aqui vão algumas dicas que o tornam pequeno.

Uma nota sobre a array A$, definida na linha 10. Da maneira como o programa está, ele só pode guardar dez nomes e dez telefones. Com memória adicional, pode guardar muito mais. Se você quiser incluir ruas nos endereços, deve aumentar o tamanho de cada elemento da array.

O primeiro ponto interessante é a maneira pela qual o computador escolhe a rotina de ação. Observe a linha 60:

```
60 GOTO CODE B$ * 100
```

Todos os computadores usam números para representar letras. Esta representação interna é chamada *código*. **CODE** é uma função que retorna o código de uma seqüência alfanumérica. **CODE** retorna apenas o primeiro caractere de uma string dada. A variável B$ é dimensionada 1 para conservar, já que nada digitado depois do primeiro caractere tem importância.

Uma vez que **GOTO** pode usar uma expressão no lugar de uma constante, a expressão usada na linha 60 direciona o programa com um simples comando. Note que não há proteção contra respostas inesperadas. Se o usuário digitar algo que não seja A, M, R ou S o programa não funcionará direito.

O resto do programa consiste de cinco rotinas, cada uma desempenhando apenas um papel. As quatro primeiras rotinas começam com números de linhas determinados pelo código do primeiro caractere da entrada de dados, pelo usuário.

A rotina para adicionar começa na linha 3800. A variável K é usada para lembrar a última parte da array usada e cada nova entrada de dados é guardada em seu primeiro espaço vazio. Quando ela está cheia, o pedido de adição é rejeitado pela linha 3820, sem aviso. Se espaço na memória não fosse problema, poderíamos dizer explicitamente ao usuário que não há mais espaços nas arrays. Como o problema existe, o usuário pode adivinhar o que aconteceu. A única pista é que o programa pára pedindo uma nova entrada de dados. Você certamente corrigirá estes detalhes quando tiver mais memória.

Na linha 3830, o número da primeira array vaga é transferido para a variável E. A linha 3840 dá um salto para a linha 6000, que é o início da quinta rotina.

A rotina na linha 6000 simplesmente adiciona uma nova entrada de dados. Esta rotina sempre usa a variável E para definir a localização da nova entrada. Se o usuário está modificando um dado, E é o número da entrada de dados a ser alterada.

A rotina de modificar começa na linha 4000. Ela, primeiro, pergunta pelo número da entrada. O usuário que quiser modificar uma entrada deve, primeiro, REVER para descobrir o número dela e então modificar seus dados.

Por fim, a rotina para REVER começa na linha 5500. Ela pergunta, primeiro, por uma *letra a procurar*; esta letra é usada para procurar uma informação específica. Neste caso o programa vai à array e compara sua letra com o primeiro caractere em cada elemento. Quando as letras casam, o programa imprime o elemento. Por exemplo, se

a sua chave procurar um J (só pode ser letra), o computador imprime todas as entradas começando por J.

A quarta rotina, simplesmente, salva o programa. Após entrar com os seus dados, posicione sua fita cassete, comece a gravação, digite S e tecle **ENTER**. Como de costume, é sempre bom ter duas ou três cópias do programa caso aconteça algo a uma das cópias. Uma vez salvo o programa pela primeira vez, o programa volta à linha 30, de modo que você só precisa teclar S e **ENTER**, sem desligar o gravador. Quando você carregar o programa, a versão carregada continuará da linha 5610 e a linha 30 será executada imediatamente.

APÊNDICE A

Acessórios (Periféricos)

Um gravador cassete é, provavelmente, o primeiro acessório que você vai desejar comprar para o seu Timex/Sinclair 1000.

Há grande número de acessórios e os que são ligados diretamente ao computador são chamados *periféricos.* Esta lista, que começa com o mais útil deles, cobre a maioria dos periféricos que você pode adquirir. Para maiores informações, leia artigos nas revistas especializadas ou lojas do ramo.

Impressoras

A impressora é um acessório muito útil pois imprime tudo que a televisão pode mostrar.

A impressora vendida pela Timex usa rolos de papel sensível ao calor e com largura de 3 1/2 polegadas. Há impressoras similares de outras companhias.

Instalar uma impressora é simples; basta ligar o plug da impressora no slot da parte de trás de seu Timex/Sinclair 1000.

Há três comandos de impressão. **LPRINT** que trabalha como o comando **PRINT**. Para fazer com que seu programa seja impresso basta substituir **PRINT** por **LPRINT**.

O segundo comando, **LLIST**, funciona exatamente como o comando **LIST**, só que usando a impressora em lugar da televisão.

O terceiro comando, **COPY**, copia tudo que estiver na tela da televisão para a impressora.

Expansões de Memória

Seu Timex/Sinclair 1000 vem com pouco mais que 2000 bytes de memória. Um *byte* é uma pequena informação correspondente a um caractere. Mil bytes é equivalente a um kbyte ou, simplesmente, 1 K. O Timex/Sinclair 1000 vem com 2K de memória.

À medida que você usa esta máquina, rapidamente descobrirá que 2000 caracteres não é uma grande memória. Você ficará sem memória se fizer um programa muito

---

NOTA: Se você usar **LPRINT** e a impressora não estiver conectada, o computador não funcionará. Se isto acontecer, basta pressionar a tecla **BREAK**.

grande ou um programa que guarde informações. O caderno de telefones no Capítulo 11, por exemplo, usará rapidamente os 2K de memória.

É possível comprar uma expansão de 16K da Timex ou de outros fabricantes.

Não há considerações especiais para usar uma memória expandida. O computador poderá lidar com mais linhas de programa. Com a memória adicional, você não precisará observar tanto o tamanho de seus programas. Isto significa que eles podem ser entendidos mais facilmente. Por exemplo, o comando **REM**, ignorado neste livro, pode ser usado para escrever notas em seu programa. Linhas com o comando **REM** são ignoradas pelo computador e são usadas para auxiliar o usuário. Os programas deste livro não usam comando **REM**, pois eles se tornam um luxo quando se tem pouca memória. Em programas maiores, **REM** torna-se uma necessidade.

Com memória adicional é possível usar nomes maiores para variáveis. Para economizar espaço, as variáveis neste livro têm apenas uma letra e para elas, com maior memória é possível usar nomes mais significativos.

O limite máximo de espaço que um programa pode ocupar é 16K, embora outras companhias que não a Timex, vendam expansão para 32K e 64K.

Baterias Recarregáveis

Ocasionalmente, você quererá usar o seu computador quando não houver luz. Se isto acontecer, uma bateria recarregável lhe será útil.

Uma bateria recarregável pode alimentar seu computador durante um período de trinta minutos a duas horas (dependendo do modelo). Ela poderá atuar, também, como um regulador de voltagem, evitando que o seu computador fique danificado caso a tensão (voltagem) varie bruscamente.

Interfaces de Entrada e Saída

Uma interface de entrada/saída (ou E/S) é um periférico eletrônico que controla as comunicações entre o computador e o mundo externo. Interfaces E/S possibilitam conectar seu Timex a periféricos construídos para outros computadores: outros tipos de impressoras, disk drives (ver adiante), leitoras de caracteres, leitoras de cartão, leitoras de códigos de barras e muitos outros. Você também pode usar interfaces E/S para ligar seu Timex a outros computadores.

Interfaces E/S também podem controlar as lâmpadas de uma casa, sistema de alarme ou até mesmo um robô.

Modems

Um modem (modulador/demodulador) permite que seu computador se comunique com outro computador através de uma linha telefônica. Se você possui um modem, pode obter um grande número de serviços para computadores como notícias, relatórios sobre estoque de mercadorias e "compras eletrônicas."

Interfaces para Voz e Som

Interfaces para voz e som permitem que o seu computador imite a voz humana e toque músicas através de um alto-falante. Em geral, uma interface não é capaz de produzir os dois tipos de som, já que a voz é bem diferente da música.

Interfaces para Cor

Essas interfaces são usadas com TV em cores.

Teclados Tipo Máquina de Escrever

O maior inconveniente do Timex/Sinclair 1000 é o teclado; mas há alguns, tipo máquina de escrever, que podem substituir o teclado original. Eles simplificam e facilitam o uso do computador. Mas tome cuidado! Muitos teclados são muito difíceis de serem instalados. Procure um que seja de fácil instalação.

Disk Drives e Outros Periféricos para Armazenamento em Massa de Dados

Um periférico com um gravador cassete é conhecido como um periférico para armazenamento em massa, já que guarda muito mais informações do que a memória in-

terna do computador. Um gravador cassete possui duas limitações: a primeira é a lentidão, levam-se minutos para salvar ou carregar um programa. A segunda é que você deve preparar o gravador para o uso do computador.

A maneira mais barata de resolver um desses problemas é um tipo especial de gravador, chamado string-floppy. Este tipo utiliza uma fita sem fim que está em loop constante e evita o uso da tecla de retrocesso rápido. O computador controla a gravação e a reprodução do gravador.

Um disk drive é um periférico de armazenamento muito mais rápido que guarda as informações em discos cobertos com filme magnético, similares àqueles encontrados em fitas cassete. Os disk drives, em geral, são bem caros. O mais barato usa um disco flexível (pois você consegue dobrá-lo) e tem de 3 a 8 polegadas de diâmetro.

Joysticks

Se você gosta de jogos, um joystick aumentará o seu prazer. Sendo um acessório comum em vídeo-games caseiros, também são disponíveis para seu Timex.

Leitoras Óticas de Códigos de Barra

Códigos de barras aparecem em quase tudo (em outros países), desde livros até doces. Leitoras óticas são usadas para entrada de dados.

Interfaces de Expansão

Se você está interessado em desenvolver seu próprio hardware para o Timex, seja para uso próprio ou venda, uma interface de expansão lhe será de grande utilidade. Esta interface expande o número de acesso à placa E/S, na parte de trás do seu Timex/Sinclair 1000.

Outros Periféricos [15]

Existem outros periféricos úteis com "light pens" que permitem que você desenhe na TV. Uma rápida olhada nas revistas especializadas pode dar uma idéia do que há para o seu Timex/Sinclair 1000.

---

[15] Nem todos os periféricos aqui apresentados estão à venda no Brasil.

APÊNDICE B

Anatomia do Computador

Os Computadores e os Seres Humanos

Um computador pode ser comparado a um ser humano. Ele (o computador) tem cérebro, ouvidos, maneiras de se comunicar, um coração e um aparelho circulatório.

O cérebro humano é composto de duas partes principais: a que pensa e a que se lembra das coisas. O cérebro do Timex/Sinclair 1000 tem um microprocessador Z80A para calcular; quando você manda o computador somar dois números, é o microprocessador que faz o trabalho. O microprocessador não seria bom se não pudesse se lembrar dos números a serem somados; por isso, também tem memória.

O computador tem meios de se comunicar com você. Diga, pelo teclado, o que o computador deve fazer: quando você digita algo, o computador processa sua mensagem. Se ligá-lo a uma televisão, ele mostrará suas mensagens e as dele próprio.

O aparelho circulatório do computador bombeia eletricidade. A fonte de alimentação do Timex/Sinclar 1000 "pega" a eletricidade da tomada de sua casa e converte-a para uso do computador.

Qualquer computador, pequeno ou grande, tem as mesmas cinco partes essenciais: um processador que pensa, uma memória que guarda as informações para o processador, um teclado, uma tela e uma fonte de alimentação.

Acessórios

Qualquer coisa colocada na memória de seu Timex/Sinclair 1000 ficará lá, até que se desligue o computador. Um gravador cassete elimina o trabalho de redigitar os programas. O Capítulo 5 explicou como usar o gravador com seu Timex/Sinclair 1000.

Um gravador cassete é um acessório essencial. Uma impressora também é útil para manter uma cópia permanente de dados.

Há, também, expansões de memória para seu Timex/Sinclair 1000. Quando você grava seus programas em fita cassete, pode recarregá-los para a memória do computador, embora este tenha sempre uma quantidade

limitada de memória. Você pode aumentá-la com uma expansão de memória.

Hardware e Software

O que descrevemos até agora — as partes físicas de seu computador, as partes que você pode tocar — são chamadas hardware do sistema.

Uma simples calculadora de bolso é feita apenas de hardware. Teoricamente, você pode abrir sua calculadora e tocar as partes que somam, dividem etc.

Um programa de computador, por outro lado, é intocável. Você não pode abrir seu computador e tocar em um pedaço do "programa para caderno de telefones". Isto acontece porque os computadores seguem passos que dizem o que eles devem fazer. Um passo seria "pegue o nome da pessoa", seguido de outro que seria "pegue o telefone".

Essas seqüências de passos são programas. Os programas são chamados de *software* para mostrar que são diferentes do hardware do qual o computador é feito, isto é, o hardware é a parte física do computador.

É o software que realmente diferencia o computador de uma máquina de calcular. Uma calculadora é capaz de executar tarefas prefixadas. Um computador é capaz de executar muitas tarefas diferentes. Quando se quer que o computador faça algo novo, basta carregar um novo software (desde que o hardware seja adequado).

Todo computador tem um programa ou grupo de programas chamados de sistema operacional. Quando você usa o seu Timex/Sinclair 1000 para fazer contas, o *sistema operacional* diz ao hardware o que fazer. Quando você digita algo, as letras aparecem na tela porque o sistema operacional as põe lá.

Um Pouco Mais Sobre Memória

Embora seja o software que faz seu computador funcionar, é o hardware que determina o quanto o software pode fazê-lo. Você pode, por exemplo, fazer um programa que permita que seu computador ande (se conseguir, patenteie-o rápido), mas seu Timex/Sinclair 1000 não pode fazê-lo, pois não existe ainda o hardware necessário. Analogamente, existe um limite de instruções que você pode colocar na memória de seu Timex/Sinclair 1000.

O sistema operacional que vem com o computador está permanentemente trancado em um determinado local, na memória do mesmo. Você não pode alterá-lo, movê-lo ou danificá-lo. Ele é mantido em um tipo especial de memória que pode apenas ser lida. Este tipo de memória é chamada "*read only memory*" ou **ROM.** Você deve conhecer algum vídeo-game que utiliza cartuchos. Se quiser um jogo diferente, deve trocar o cartucho. Na verdade, os cartuchos contêm **ROM.** O vídeo-game é um computador especial e a **ROM** no cartucho é um programa.

Se o seu Timex/Sinclair 1000 só tivesse **ROM** não seria muito melhor do que uma máquina à base de cartucho.O que faz do seu Timex/Sinclair 1000 um computador versátil é a memória que pode ser modificada. Você pode fazer seus programas, digitar programas de livros ou revistas ou mesmo comprá-los.

Este tipo de memória chama-se "*random access memory*" (memória de acesso randômico) ou **RAM.** Ela é assim chamada porque pode-se ler ou escrever nela. Além do mais, você não tem que seguir o programa passo a passo. A qualquer momento pode acessar qualquer parte do programa. No mais, você pode se mover *randomicamente* pela memória que guarda o programa.

APÊNDICE C

Glossário

ARGUMENTO. O argumento de uma função é o valor com o qual ela opera para produzir o resultado desejado. Por exemplo: se você digitou **PRINT-PEEK-1200** você está dizendo ao computador para procurar na locação da memória de endereço 1200 e mostrar o que está lá. Neste caso, **PEEK** é a função e 1200 é o argumento.

ARRAY. (Variável Indexada-Matriz). É um grupo de variáveis (chamadas elementos) que têm o mesmo nome. Você pode criar todos os elementos de uma array com um comando **DIM** e acessá-los com o nome de variável seguida do índice.

BASIC. Beginner's All-purpose Symbolic Instruction Co-

de. (Código Simbólico Para Todos os Fins, Para Principiantes). É a linguagem usada em seu T/S 1000.

BUG. É um erro em seu programa e que não permite sua execução.

BYTE. Um byte representa uma informação de um caractere e consiste de oito dígitos binários ou bits.

CARACTERES INVERSOS. Normalmente, o seu computador imprime as letras pretas e o fundo é branco. No modo gráfico, entretanto, você pode ter caracteres inversos, isto é, letras brancas com fundo preto.

CÓDIGOS DE ERROS. São números que aparecem no canto inferior esquerdo da tela, separados por uma barra, indicando o motivo da suspensão da execução do programa.

CONSTANTE. É um item cujo valor permanece fixo durante a execução do programa. Exemplo: números.

CONSTANTE STRING. (Veja *string*).

CONTADOR DO LOOP. É uma variável numérica cujo valor é incrementado ou decrementado de um certo número, cada vez que o loop é executado.

CURSOR. O cursor, quando aparece no canto inferior esquerdo da tela, é a maneira de o computador dizer que está pronto para receber seus comandos.

F **CURSOR.** É acessado pressionando-se **SHIFT/ENTER.** Quando o cursor F aparece, o cursor espera que

a próxima entrada seja uma função e depois o cursor F transforma-se no cursor L .

G CURSOR. É acessado pressionando-se **SHIFT/9**. Quando o cursor G aparece, o computador aceita caracteres gráficos. O cursor G é removido pressionando-se **SHIFT/9** de novo.

K CURSOR. Aparece no início de cada linha de programa. É um sinal de que o computador espera uma palavra-chave. Depois que a palavra-chave é digitada o cursor L aparece.

L CURSOR. Quando o cursor L aparece na tela, você poderá digitar letras, números e espaços.

DEBUG. Achar e eliminar erros em um programa. É importante para o sucesso do programa.

ECO. Um caractere que aparece instantaneamente na tela quando você digita, chama-se eco.

EXPONENCIAÇÃO. É o processo de elevar um número a uma potência. Exemplo 10 * 10 * 10* 10 * 10 pode ser escrito como 10 ** 5 (dez elevado à quinta potência). Expressar números altos usando exponenciação economiza espaço na memória.

EXPRESSÃO CONDICIONAL. É uma cláusula com um comando que age de uma maneira se for verdadeira ou de outra se for falsa. No comando **IF ... THEN, IF** é a cláusula condicional e **THEN** é a porção que será executada

se for verdadeira. Se for falso o programa continua na próxima linha.

EXPRESSÃO. (Veja *expressão numérica*).

EXPRESSÃO NUMÉRICA. São dois ou mais números relacionados por um operador.

EXPRESSÃO RELACIONAL. É uma expressão matemática para relacionar dois itens através de um operador relacional. São operadores relacionais: > = (maior ou igual a); < = (menor ou igual a); > (maior que); < (menor que); = (igual) e <> (diferente).

EXPRESSÃO STRING. São strings conectadas pelo operador + (adição). Expressões strings podem conter constantes ou variáveis string. Exemplo: **LET**-A$ = "ABC" + B$. é uma expressão string válida.

FUNÇÕES. São programas especiais guardados na **ROM** do computador, que executam operações matemáticas. Podem ser acessadas quando o cursor **F** é mostrado na tela.

INTEIRO. É um número expresso sem frações ou casas decimais. Exemplo: 0 (zero), −5, 1, 7, 42, − 13.

K (Kilo ou Mil). Na terminologia do computador 1K é 1024 bytes. O T/S 1000 tem uma **RAM** de 2K ou 2048 bytes e uma **ROM** de 8K ou 8192 bytes.

LINHA CORRENTE. Em uma listagem de um programa é a linha que contém o pointer da linha corrente. Pressio-

nando-se **SHIFT/EDIT**, a linha corrente pode ser copiada na parte inferior da tela onde pode ser editada quando se quiser.

LINHA DE COMANDO. A linha de programa que você cria na parte inferior da tela é considerada uma linha de comando até que você tecle **ENTER.**

LINHA DE PROGRAMA. Uma linha com comandos precedida por um número torna-se uma linha de programa quando você tecla **ENTER.** As linhas de programa somente são executadas quando se dá o comando **RUN.**

LITERAL STRING. (Veja *string*).

LOOP INFINITO. Em um programa, é um loop que, uma vez iniciado, não acaba mais até que se suspenda a execução do programa.

LOOP INTERNO. É um loop que está totalmente dentro de outro loop.

MODO DE COMANDO. Quando você digita um comando e **ENTER,** sem um número de linha, é considerado um comando direto, pois é executado imediatamente. Isto é chamado modo de comando ou modo imediato. Estes comandos não entram na listagem do programa que estiver na memória.

MODO GRÁFICO. O modo gráfico permite que você tenha caracteres invertidos bem como caracteres gráficos. O modo gráfico pode ser acessado pressionando-se

SHIFT/9 e só se pode sair pressionando **SHIFT/9.** No modo gráfico o cursor G é mostrado.

OPERADOR. É qualquer um dos cinco operadores usados em expressões numéricas: + (adição); – (subtração); * (multiplicação); / (divisão); ** (exponenciação).

PALAVRAS-CHAVE. Uma palavra-chave diz ao computador que tipo de comando você está lhe dando. Uma palavra-chave é esperada no início de cada linha de programa e só pode ser digitada com o cursor K .

POSIÇÃO DE IMPRESSÃO. É a localização do próximo item a ser impresso. Pode ser controlada por **TAB, AT, PLOT** e **UNPLOT.**

PROGRAMA. É uma série de instruções que levam o computador a realizar uma tarefa.

RAM. RANDOM ACCESS MEMORY. Memória de Acesso Randômico. É um tipo de memória no qual se pode alterar, ler e escrever dados. O conteúdo da **RAM** altera-se constantemente quando um programa é executado. É também conhecido como memória para ler e escrever.

ROM — READ ONLY MEMORY. É a memória pré-programada da qual se pode ler dados mas não se pode alterá-la ou escrevê-la.

STRING — CADEIA ALFANUMÉRICA. Qualquer coisa que esteja entre aspas. Quando você imprime uma string, as aspas desaparecem. Para usar as aspas use **SHIFT/Q.**

STRING LITERAL — Veja *string.*

VALOR ABSOLUTO. É o valor positivo de um número, sem o sinal. Por exemplo: o valor absoluto de − 5 é 5 e o de + 5 é 5.

VARIÁVEL. É um nome simbólico que pode assumir diferentes valores durante a execução de um programa.

VARIÁVEL DO LOOP. Veja *contador do loop.*

VARIÁVEL NUMÉRICA. É uma variável usada para conter apenas informações numéricas. Os nomes das variáveis numéricas podem ter qualquer tamanho e devem começar por uma letra.

VARIÁVEL STRING. É uma variável usada para guardar uma string. Seu nome deve ter uma única letra seguida de um cifrão. Exemplos: C$, B$, E$, Z$, etc.

VARIÁVEL STRING DIMENSIONAL. É o nome de uma variável usada em uma array alfanumérica (string), a qual deve ser definida como uma variável string seguida de parênteses com o número de strings e o tamanho de cada uma delas. Exemplo: **DIM**-A$ (10, 8). Veja também *array.*

APÊNDICE D

Códigos de Erro

Toda vez que seu Timex/Sinclair 1000 executar um comando ou suspender a execução de um programa, aparecerá uma mensagem na parte inferior da tela do tipo:

número 1/número 2

Número 2 é um número de linha. Se esta mensagem aparecer após um comando no modo imediato, este número é sempre 0 (zero). Se a mensagem aparecer após a execução de um programa, este número é o número da última linha executada.

Número 1 é o número que indica por que a execução do comando ou do programa foi suspensa.

Um código de número 0 (zero) indica que a operação foi bem sucedida. Se você tivesse executado um programa, o 0 (zero) indicaria que o programa acabou. Se você suspender a execução de um programa com um comando **STOP**, terá um código 9 ou, alguma vezes, D. Se usar a tecla **BREAK**, será um código D.

Outros códigos indicam que o computador teve algum tipo de problema na tentativa de executar seu programa.

Há um total de 15 códigos: 0 - 9, A, B, C, D e F (não há código E). Aqui estão os códigos e seus significados:

CÓDIGO	SITUAÇÃO	SIGNIFICADO
0	QUALQUER	— Execução bem sucedida ou salto para uma linha de maior número do que qualquer outra existente. Uma indicação 0 não altera o número da linha usada por **CONT**.
1	NEXT	— A variável de controle não existe (não foi estabelecida por uma instrução **FOR**), mas existe uma outra variável com o mesmo nome.
2	QUALQUER	— Foi utilizada uma variável indefinida. No caso de uma variável simples, isso ocorrerá se a mesma for usada antes de ter sido referenciada por um **LET**. Para variáveis subscritas, se as mesmas forem usadas sem antes serem dimensionadas pela instrução **DIM**. E para uma variavél de controle isso ocorrerá caso a mesma seja usada antes de ter sido definida como variável do loop por uma instrução **FOR** e se não houver nenhuma variável simples com o mesmo nome.
3	VARIÁVEIS SUBSCRITAS	— Subscrito fora de faixa. Caso o subscrito (matriz) esteja fora da faixa (ou seja, negativo ou acima de 65535), ocorrerá o erro B.

CÓDIGO	SITUAÇÃO	SIGNIFICADO
4	**LET, INPUT, DIM, PRINT, LIST, PLOT, UNPLOT, FOR, GOSUB; ÀS VEZES DURANTE AVALIAÇÃO DE FUNÇÕES**	— Espaço insuficiente na memória. Observe o número de linha na indicação (após 0/) poderá estar incompleto na tela, devido à falta de memória; assim, por exemplo, 4/20 poderá surgir como 4/2.
5	**PRINT, LIST**	— Não há mais espaço na tela. **CONT** criará espaço limpando a tela.
6	**QUALQUER CÁLCULO ARITMÉTICO**	— Sobrecarga aritmética: os cálculos produziram um número superior a 10^{38}.
7	**RETURN**	— **RETUNR** sem um **GOSUB** correspondente.
8	**INPUT**	— Você tentou um comando **INPUT** não permitido.
9	**STOP**	— Instrução **STOP** executada. **CONT** não tentará reexecutar a instrução **STOP**.
A	**SQR, LN, ASN, ACS**	— Argumento inválido para certas funções.
B	**RUN, RAND, POKE, DIM, GOTO, GOSUB, LIST, PAUSE, PLOT, UNPLOT, CHR$, PEEK, USR, LLIST. Acesso a array**	— Número inteiro fora da faixa. Quando um número é requerido, o argumento de ponto flutuante é arredondado para o número inteiro mais próximo. Caso isto esteja fora de uma faixa adequada, surge erro B.
C	**VAL**	— O texto do argumento (de string) não forma uma expressão numérica válida.
D	**AO FIM DE QUALQUER INSTRUÇÃO OU EM LOAD, SAVE, LPRINT, LLIST OU COPY**	— 1. Programa interrompido por **BREAK**. 2. A linha **INPUT** começa com o **STOP**.
E		— Não utilizado.
F	**SAVE**	— O nome fornecido para o programa é uma string vazia.

APÊNDICE E

Como Eliminar Dificuldades

O mau funcionamento do computador pode aparecer sob várias formas. O computador pode se recusar a mostrar sua mensagem inicial, pode se recusar a mostrar os caracteres e palavras que você digitar e também pode se recusar a aceitar seus comandos. Aqui vão algumas sugestões para localizar o problema.

O computador está corretamente instalado? Verifique as instruções do Capítulo 1 deste livro. Se o cursor apareceu na tela, este não é o problema. Se a tela estiver em branco (toda cinza), é possível que o vertical de sua televisão esteja desregulado.

A fonte de alimentação está ligada a uma tomada em boas condições? Talvez algum fusível se tenha queima-

do. Pode parecer inacreditável, mas muitos chamados a firmas de reparo de computadores foram devidos a fusíveis queimados e tomadas não conectadas.

Há algo errado com a TV? Tente conectar seu computador a outra televisão.

O computador está superaquecido? Se você expuser o computador diretamente à luz do sol, ele pode parar de funcionar.

Há algum tipo de interferência? É possível (embora pouco provável) que sinais de rádio venham a interferir em seu computador. Há alguma antena de rádio, faixa-do-cidadão ou comercial próxima? Tente levar seu computador para outro lugar da casa.

Há algum cabo mal conectado? A melhor pista é a operação intermitente. Se seu computador parece funcionar bem e pára de funcionar quando você o move um pouco, os cabos podem estar mal conectados. Se seu computador sempre se recusar a funcionar, pode também ser os cabos.

Não ficou bom? Tente entrar em contato com a loja onde você o comprou e peça sugestões. Se ainda não funcionar leve o computador a uma oficina especializada.

ÍNDICE REMISSIVO